МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Уманський державний педагогічний університет імені Павла Тичини

ПРОГРАМУВАННЯ

Навчальний посібник

Укладачі: М. С. Ковтанюк, Л. О. Тітова

Умань
Візаві
2023

УДК 004.42(075.8)
П78

Рецензенти:

Морозов А. В., кандидат технічних наук, доцент, проректор з науково-педагогічної роботи Державного університету «Житомирська політехніка»;

Ковальов Л. Є., кандидат фізико-математичних наук, доцент, доцент кафедри математики і фізики Уманського національного університету садівництва;

Ткачук Г. В., доктор педагогічних наук, професор, професор кафедри інформатики і інформаційно-комунікаційних технологій Уманського державного педагогічного університету імені Павла Тичини.

Рекомендовано до друку вченою радою
факультету фізики, математики та інформатики
Уманського державного педагогічного університету імені Павла Тичини
(протокол №4 від 7 листопада 2023 р.)

Програмування : навч. посіб. / МОН України, Уманський держ. пед.
П78 ун-т імені Павла Тичини ; уклад.: М. С. Ковтанюк , Л. О. Тітова. – Умань : Візаві, 2023. – 186 с.

Навчальний посібник присвячено основам алгоритмізації та програмування. Висвітлені окремі характеристики мов програмування низького та високого рівнів. Особливу увагу приділено програмуванню у середовищі Scratch та програмуванню на мові Python. Теоретичний матеріал ілюструється великою кількістю практичних прикладів з детальним поясненням, наведені контрольні запитання та практичні завдання для самостійного опрацювання.

Призначений для студентів, що навчаються за ОПП «Середня освіта (Інформатика)», «Середня освіта (Фізика. Інформатика)», «Середня освіта (Математика. Інформатика)» на освітньому ступені «бакалавр» усіх форм навчання, здобувачів закладів загальної середньої освіти, викладачів та вчителів. Навчальний посібник укладено відповідно до навчальної програми дисципліни «Програмування».

УДК 004.42(075.8)

Зміст

Передмова

В основу пропонованого навчального посібника покладено цикл занять, проведених для здобувачів освіти факультету фізики, математики та інформатики Уманського державного педагогічного університету імені Павла Тичини, що навчаються за освітньо-професійними програмами «Середня освіта (Інформатика)», «Середня освіта (Фізика. Інформатика)», «Середня освіта (Математика. Інформатика)» на освітньому ступені «бакалавр». Посібник призначений для здобувачів закладів загальної середньої та вищої освіти, викладачів та вчителів.

Програмування – це одна з найбільш важливих та цікавих галузей інформатики. Вона відкриває перед Вами світ незмірних можливостей та творчих досягнень. У цьому посібнику ми намагалися об’єднати теоретичні основи програмування з практичними завданнями, які допоможуть Вам не лише зрозуміти концепції, але й навчитися застосовувати їх на практиці. Ми віримо, що практичний досвід – це найкращий спосіб засвоєння програмування.

Цей посібник охоплює широкий спектр тем, від основних понять програмування до більш складних алгоритмів та структур даних. Ви дізнаєтеся, як програмувати мовою Python та створювати ігрові проєкти у Scratch.

Ми сподіваємось, що цей посібник допоможе Вам розвинути навички програмування, підвищити рівень кваліфікації та підготувати Вас до викликів, які може поставити перед Вами сучасний світ інформатики та технологій. Програмування – це інструмент, який відкриває безмежні можливості для творчого вираження та розв’язання складних завдань.

Нехай цей навчальний посібник стане для вас важливим джерелом знань та натхнення. Бажаємо Вам захоплюючого досвіду у світі програмування!

Розділ 1. АЛГОРИТМІЧНІ МОВИ ТА СИСТЕМИ ПРОГРАМУВАННЯ

1.1. Алгоритми та мови програмування

Особливе місце при розв'язанні будь-якого завдання займає розробка чи вибір алгоритму роботи. Поняття «алгоритм» широко використовується у повсякденному житті, проте частіше у математиці та програмуванні.

Термін алгоритм походить від слова *algorithmi*, що означає латинську форму запису імені відомого перського математика, географа, історика та астронома Аль-Хорезмі (799-846 рр.), який вперше виділив алгебру, як самостійну дисципліну [6].

У загальному розумінні, *алгоритм* – це набір кроків для виконання завдання [13]. Проте, більш точне визначення поняття *«алгоритм»* характеризує його як чітке описання скінченної послідовності дій, які необхідно виконати для розв'язання задачі.

Практичне розв'язування будь-якої задачі із заданими початковими даними вимагає отримання результату, тому доцільним є ще одне визначення алгоритму. *Алгоритм* – це описання послідовного процесу перетворення початкових даних на результат [30].

Кожний алгоритм пишеться під конкретну задачу з розрахунком на конкретного *виконавця*. Виконавцями алгоритмів можуть бути людина, машина, тобто ті, хто розуміє та може виконати вказівки алгоритму. *Система команд виконавця* представляє собою набір команд, які можуть бути виконані ним; кожна команда алгоритму повинна входити до системи команд виконавця. Кожну дію або команду виконавець здійснює згідно з інструкцією.

Алгоритми володіють наступними *властивостями*:

1. *Дискретність*: алгоритм розв'язку повинен складатися з послідовності окремих кроків, що відокремлені один від одного командами (вказівками), кожна з яких повинна виконується за скінчений час. Іншими словами, закінчивши виконання однієї команди, виконавець переходить до наступної.

2. *Визначеність*: кожна команда алгоритму визначає дії виконавця однозначно, і не допускає подвійного тлумачення. Тобто, має місце чітко визначений порядок виконання команд.

3. *Формальність*: будь-який виконавець, який володіє заданою системою команд, може виконати алгоритм, не вникаючи в зміст поставленої задачі.

4. *Результативність*: виконання алгоритму повинно завжди вести до результату і не може закінчуватися невизначеністю або не закінчуватися зовсім (зацикленість). Виконання будь-який алгоритму, при допустимих початкових даних за кінцеве число кроків, повинно завжди вести до результату.

5. *Масовість*: алгоритм повинен передбачати можливість зміни початкових (вхідних) умов чи то даних, а у деяких допустимих межах і можливість його використання для розв'язання типових задач (універсальність алгоритму) [6].

Існує чотири *способи представлення алгоритмів*:

- вербальний (словесний);
- алгебраїчний (за допомогою літерно-цифрових позначень);
- графічний (за допомогою блок-схем);
- за допомогою мов програмування [31].

Для подання алгоритмів у вигляді, який розуміє комп'ютер, використовують *мови програмування*. Спочатку розробляється алгоритм дій, потім алгоритм записується на одній з таких мов. У результаті отримують *текст програми* – повний, закінчений і детальний опис алгоритму мовою програмування. Текст програми переводиться в машинний код або виконується під управлінням спеціальних службових програм, які називаються трансляторами.

Самому написати програму в машинному коді досить складно, причому складність різко зростає зі збільшенням розміру програми й трудомісткістю рішення потрібного завдання. Умовно можна вважати, що машинний код прийнятний, якщо розмір програми не перевищує декількох десятків байтів і нема потреби в операціях ручного уведення/виведення даних. Сьогодні

практично всі програми створюються за допомогою мов програмування. Теоретично програму можна написати засобами звичайної людської (природної) мови – це називається програмуванням на *метамові* (подібний підхід зазвичай використовується на етапі складання алгоритму), але автоматично перевести таку програму в машинний код поки неможливо, через високу неоднозначність природної мови.

Мови програмування – штучні мови. Від природних вони відрізняються обмеженим числом «слів», значення яких зрозуміло транслятору, і дуже суворими правилами запису команд (*операторів*). Сукупність подібних вимог утворює *синтаксис* мови програмування, а *зміст* кожної команди й інших конструкцій мови – її *семантику*.

Порушення форми запису програми приводить до того, що транслятор не зрозуміє призначення оператору й видає повідомлення про синтаксичну помилку, а правильно написане, але не відповідаюче алгоритму використання команд мови приводить до семантичних помилок (логічних помилок або помилок часу виконання). Процес пошуку помилок у програмі називається *тестуванням*, процес усунення помилок – *налагодженням* [20].

1.2. Компілятори й інтерпретатори

Мова програмування – це штучна мова, яка створена для передачі команд машинам, зокрема комп'ютерам.

Хоча процесор комп'ютера розуміє тільки машинну мову, людям не зручно писати програми безпосередньо на ній. Така програма може мати тисячі та навіть мільйони бінарних інструкцій, написання яких може стати досить обтяжливим процесом. З цієї причини була створена мова програмування *асемблера*, як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, асемблер використовує короткі слова – *мнемокоди* (Рис.1.2.1).

Рисунок 1.2.1. Схема роботи асемблера

У зв'язку з тим, що мова асемблера близька за своєю природою до машинної мови, вона є мовою *низького рівня* [6].

Проте, більшість програм пишеться на *високорівневих алгоритмічних мовах програмування*. Такі мови зазвичай мають складний синтаксис, використовують слова-оператори близькі до людської мови і – що найголовніше – реалізують алгоритмічні структури для простого і зрозумілого запису програми.

З іншого боку, для виконання комп'ютером така програма має бути перетворена на машинний код або хоча б переписана низькорівневою мовою (а далі вже асемблер забезпечить її розуміння машиною). Причому команди високорівневої мови програмування можуть бути досить складними і відповідати кільком або навіть кільком десяткам машинних команд. Цей процес перетворення називається *трансляцією*, а програми, які його виконують – *трансляторами*.

Існує два типи трансляторів, що перетворюють вихідний код програм в машинні команди: *інтерпретатори* та *компілятори* [31].

Інтерпретатор бере черговий оператор мови з тексту програми, аналізує його структуру й потім відразу виконує (звичайно після аналізу оператор транслюється в деяке проміжне подання або навіть машинний код для більш ефективного подальшого виконання). Тільки після того як поточний оператор успішно виконаний, інтерпретатор перейде до наступного. При цьому якщо той самий оператор повинен виконуватися в програмі багаторазово, інтерпретатор щораз буде виконувати його так, начебто зустрів уперше. Внаслідок цього,

програми, у яких потрібно здійснювати великий обсяг повторюваних обчислень, можуть працювати повільно. Крім того, для виконання такої програми на іншому комп'ютері там також повинен бути встановлений інтерпретатор – адже без нього текст програми є просто набором символів.

Компілятори повністю обробляють весь текст програми (він іноді називається вихідний код). Вони переглядають його в пошуках синтаксичних помилок (іноді кілька разів), виконують певний значеннєвий аналіз і потім автоматично переводять (транслюють) на машинну мову – генерують машинний код. Нерідко при цьому виконується оптимізація за допомогою набору методів, що дозволяють підвищити швидкодію програми (наприклад, за допомогою інструкцій, орієнтованих на конкретний процесор, шляхом виключення непотрібних команд, проміжних обчислень і т.д.). У результаті закінчена програма виходить більш компактною та ефективною, працює в сотні разів швидше програми, виконуваної за допомогою інтерпретатора, і може бути перенесена на інші комп'ютери із процесором, що підтримує відповідний машинний код.

Основний недолік компіляторів – трудомісткість трансляції мов програмування, орієнтованих на обробку даних складної структури, часто заздалегідь невідомої або динамічно мінливої під час роботи програми. Тоді в машинний код доводиться вставляти безліч додаткових перевірок, аналізувати наявність ресурсів операційної системи, динамічно їх захоплювати й звільняти, формувати й обробляти в пам'яті комп'ютера складні об'єкти, що на рівні жорстко заданих машинних інструкцій здійснити досить важко, а для ряду завдань практично неможливо.

За допомогою інтерпретатора, навпаки, припустимо в будь-який момент припинити роботу програми, дослідити вміст пам'яті, організувати діалог з користувачем, виконати як завгодно складні перетворення даних і при цьому постійно контролювати стан навколишнього програмно-апаратного середовища, завдяки чому досягається висока надійність роботи. Інтерпретатор при виконанні кожного оператора перевіряє безліч характеристик операційної

системи й при необхідності максимально докладно інформує розроблювача про виникаючі проблеми. Крім того, інтерпретатор дуже зручний для використання в якості інструменту вивчення програмування, тому що дозволяє зрозуміти принципи роботи будь-якого окремого оператора мови.

У реальних системах програмування перемішані технології і компіляції, і інтерпретації. У процесі налагодження програма може виконуватися по кроках, а результуючий код не обов'язково буде машинним – він навіть може бути вихідним кодом, написаним на іншій мові програмування (це істотно спрощує процес трансляції, але вимагає компілятора для кінцевої мови), або проміжним машинно-незалежним кодом абстрактного процесора, що у різних комп'ютерних архітектурах стане виконуватися за допомогою інтерпретатора або компілюватися у відповідний машинний код [20].

1.3. Еволюція та класифікація мов програмування

Розвиток мов програмування бере свій початок із XIX століття. Тоді англійський учений Чарльз Беббідж створив механічну обчислювальну машину, програму для якої розробила Ада Лавлейс, донька лорда Байрона (на честь цієї жінки названо мову програмування ADA). Однак мови програмування в сучасному розумінні фактично почали розвиватися з появою електронно-обчислювальної машини (ЕОМ).

Мова програмування – це алгоритмічна мова, призначена для запису алгоритму й даних для ЕОМ.

Сьогодні нараховуються сотні різних мов програмування і їх модифікацій, проте лише окремі з них набули широкого розповсюдження. На різних етапах розвитку ЕОМ популярними були такі мови програмування, як Fortran, «Кобол», Algol-60, PL-1, Algol-68, ADA, C, Basic, Pascal, Prolog, Delphi та ін. Найпотужнішими є мови C++, C# і Java.

Існує значна кількість різноманітних класифікацій мов програмування. Однак до основних ознак належать ступінь залежності мови від апаратних

засобів, принципи програмування й орієнтація на клас задач. На рис. 1.3.1. наведено схему класифікації мов програмування за цими ознаками [25].

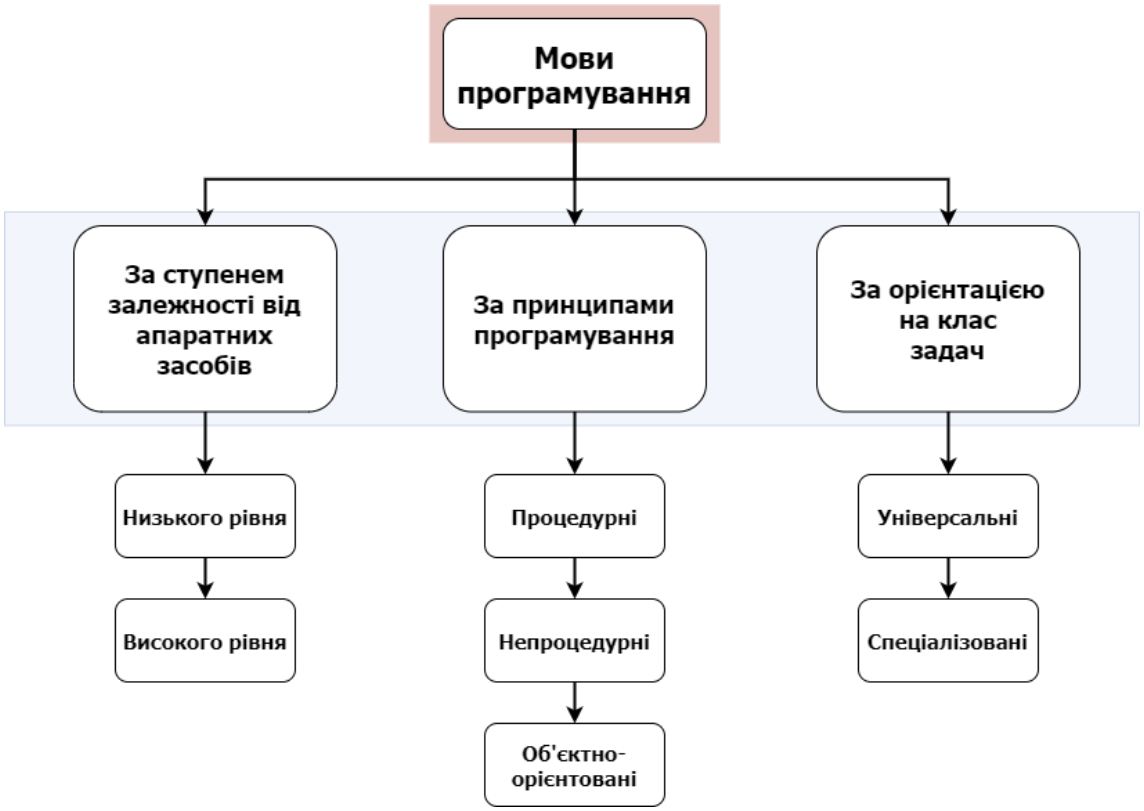


Рисунок 1.3.1. Класифікація мов програмування

Незалежно від того, якою мовою розроблено програму, якою мовою розроблено програму, в комп’ютері виконується лише програма машинною мовою, тобто мовою, що подається двійковими символами 0 і 1. Для перших ЕОМ програмісти власноруч розробляли програми машинними мовами. Але це була досить складна робота, яка також потребувала гарного знання структури та принципів взаємодії пристроїв комп’ютера. Пізніше з’явились мови символьного кодування (МСК), у яких команди подавалися не двійковим кодом, а символами.

Зрозуміло, що розроблення програм МСК полегшувало роботу програміста, адже, перетворення символьного коду в машинні команди комп’ютер виконує автоматично.

Потім до МСК почали включати *макрокоманди*, які реалізуються послідовністю з кількох машинних команд. Використання макрокоманд у МСК

ще більше полегшило розробку програм. Різновиди мов символьного кодування називають *автокодом* і *мовою асемблера*.

Проте слід зазначити, що мови символьного кодування залишалися ще надто складними. Вони доступні здебільшого професійним програмістам, оскільки програмування ними потребує спеціальних знань, у тому числі знання структурних особливостей ЕОМ. Такі мови отримали назву *машинно-орієнтованих*, або *мов низького рівня*. Практично кожен тип комп'ютера мав власну мову програмування низького рівня, що обмежувало обмін програмами. Системні програмісти використовували такі мови для розробки системного та прикладного програмного забезпечення ЕОМ.

Водночас уже на перших етапах розвитку обчислювальної техніки починається розроблення мов програмування, які були б доступними для широкого кола користувачів і не були пов'язані з конкретною обчислювальною машиною, їх назвали *мовами високого рівня*. Першою такою мовою, яка набула широкого визнання серед програмістів, була мова Fortran (Фортран). Вона була розроблена 1954 року в США. Ця мова близька до звичайної мови алгебри й орієнтована на розв'язування обчислювальних задач. Група науковців різних країн 1960 року розробила мову програмування Algol-60, яка теж орієнтована на розв'язування обчислювальних задач. Отже, за ступенем залежності від апаратних засобів розрізняють *мови низького та високого рівнів*.

Із розвитком і вдосконаленням обчислювальних машин розширювалися й галузі їх використання. Природно, що вдосконалювалися й розвивалися також і мови програмування. Їх розвиток здійснювався шляхом як спеціалізації, так і універсалізації. Однією з перших спеціалізованих була мова Cobol (Кобол), розроблена 1961 року в США й орієнтована на розв'язування економічних задач. Пізніше було розроблено десятки різних спеціалізованих мов, наприклад Simula (Симула) – мова моделювання, Lisp (Лісп) – мова для інформаційно-логічних задач, RPG (РПГ) – мова для розв'язування навчальних задач тощо.

Отже, з одного боку, потреби в розв'язуванні різноманітних задач, а з іншого – удосконалення й поява нових структур обчислювальних машин

сформували нові вимоги до мов програмування. Сутність цих вимог полягала в тому, що вони мали забезпечити успішне розв’язування різних задач людської діяльності за максимального використання можливостей ЕОМ.

Мови програмування, які певним чином відповідають зазначеним вимогам, належать до класу універсальних мов. Найвідомішими мовами цього класу були PL-1, розроблена 1964 року, й Algol-68. Однак ці мови виявилися досить складними, до того ж PL-1 не задовольняла вимогам надійності складання програм. Потім широкого розповсюдження набула універсальна мова Pascal, розроблена в Швейцарії 1973 року.

Особливою популярністю в усьому світі користувалася мова Basic, яка відрізняється своєю простотою. Вона є незамінною для розв’язування простих задач. Однак ця мова мало придатна для розв’язування складних наукових, економічних та інших задач.

Отже, за орієнтацією на клас задач мови програмування поділяють на *універсальні* та *спеціалізовані*. Універсальні мови призначені для розв’язування широкого класу задач. До них належать PL-1, Algol, Pascal, C та ін. Особливим класом універсальних мов є *мови візуального програмування* (Visual Basic, Delphi й ін.). *Спеціалізовані мови програмування* враховують специфіку предметної галузі. У наш час існують десятки спеціалізованих мов програмування, наприклад, мови вебпрограмування, мови скриптів тощо. На рис. 1.3.2 наведено класифікацію спеціалізованих мов за чотирма ознаками та деякі їхні типи.

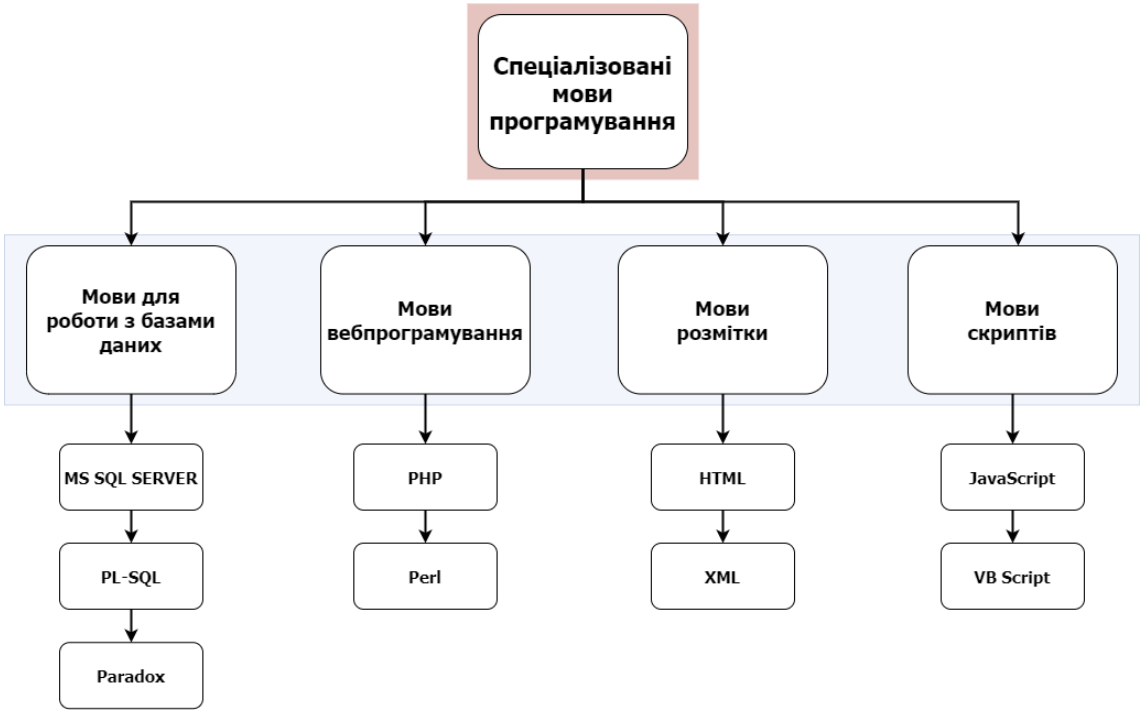


Рисунок 1.3.2. Класифікація спеціалізованих мов програмування

Мова скриптів призначена для створення невеликих допоміжних програм, наприклад, JavaScript використовується для створення динамічних об’єктів на вебсторінках. Мови розмітки містять шаблони й засоби опису вмісту, структури та формату електронних документів. Наприклад, HTML забезпечує розмітку гіпертекстового документа. Мови для роботи з базами даних забезпечують створення й супровід баз даних. Зазначимо, що не всі перелічені мови є мовами програмування в класичному розумінні. Наприклад, HTML є просто мовою розмітки тексту. Але традиційно так склалося, що і її часто називають мовою програмування.

За принципами програмування розрізняють процедурні (імперативні), непроцедурні мови та мови об’єктно-орієнтованого програмування. *Процедурні мови* основані на описі послідовної зміни стану комп’ютера, тобто значення комірок пам’яті, стану процесора й інших пристроїв. Вони маніпулюють даними в покроковому режимі, використовуючи послідовні інструкції. У процедурних мовах дотримують чіткої структуризації програм, тому їх ще називають мовами

структурного програмування. Приклади цих мов: Fortran, Algol, Pascal, Basic тощо.

Непроцедурні мови ефективні для програмування пошуку даних у великих обсягах інформації, а також програмування задач, процес розв'язування яких неможливо описати точно (переклад, розпізнавання образів). У цих мовах сама процедура пошуку розв'язання вбудована в інтерпретатор мови. До непроцедурних належать мови функціонального та логічного програмування.

Мови об'єктно-орієнтованого програмування почали розвиватися наприкінці XX століття. Вони містять конструкції, що дають змогу визначати об'єкти, які належать класам і мають властивості роботи з абстрактними типами даних. До таких мов належать C++, Object Pascal, Java, C# та ін [25].

Покоління мов програмування

Мови програмування прийнято ділити на п'ять поколінь. У перше покоління входять мови, створені на початку 50-х років, коли перші комп'ютери тільки з'явилися на світ. Це була перша мова асемблера, створена за принципом «одна інструкція – один рядок».

Розквіт другого покоління мов програмування прийшовся на кінець 50-х – початок 60-х років. Тоді був розроблений символічний асемблер, у якому з'явилося поняття змінної. Він став першою повноцінною мовою програмування. Завдяки його виникненню помітно зросли швидкість розробки й надійність програм.

Появу третього покоління мов програмування прийнято відносити до 60-м років. У цей час народилися універсальні мови високого рівня, з їхньою допомогою вдається вирішувати завдання з будь-яких галузей. Такі якості нових мов, як відносна простота, незалежність від конкретного комп'ютера й можливість використання потужних синтаксичних конструкцій, дозволили різко підвищити продуктивність праці програмістів. Зрозуміла більшості користувачів структура цих мов залучила до написання невеликих програм (як правило, інженерного або економічного характеру) значне число фахівців з

некомп’ютерних галузей. Переважна більшість мов цього покоління успішно застосовується й сьогодні.

З початку 70-х років по теперішній час триває період мов четвертого покоління. Ці мови призначені для реалізації великих проєктів, підвищення їхньої надійності й швидкості створення. Вони звичайно орієнтовані на спеціалізовані галузі застосування, де гарних результатів можна домогтися, використовуючи не універсальні, а проблемно-орієнтовані мови, що оперують конкретними поняттями вузької предметної галузі. Як правило, у ці мови вбудовуються потужні оператори, що дозволяють одним рядком описати таку функціональність, для реалізації якої на мовах молодших поколінь потрібні були б тисячі рядків вихідного коду.

Народження мов п’ятого покоління відбулося в середині 90-х років. До них відносяться також системи автоматичного створення прикладних програм за допомогою візуальних засобів розробки, без знання програмування. Головна ідея, що закладається в ці мови, – можливість автоматичного формування результуючого тексту на універсальних мовах програмування (який потім потрібно відкомпілювати). Інструкції ж уводяться в комп’ютер у максимально наочному виді за допомогою методів, найбільш зручних для людини, не знайомої із програмуванням [20].

1.4. Рівні мов програмування

Різні типи процесорів мають різні набори команд. Якщо мова програмування орієнтована на конкретний тип процесора й ураховує його особливості, то він називається *мовою програмування низького рівня*. У цьому випадку «низький рівень» не значить «поганий». Мається на увазі, що оператори мови близькі до машинного коду й орієнтовані на конкретні команди процесора.

Мовою найнижчого рівня є *мова асемблера*, що просто представляє кожну команду машинного коду, але не у вигляді чисел, а за допомогою символьних умовних позначок, які називають *мнемоніками*. Однозначне перетворення однієї машинної інструкції в одну команду асемблера називається *транслітерацією*.

Тому що набори інструкцій для кожної моделі процесора відрізняються, конкретній комп'ютерній архітектурі відповідає своя мова асемблера, і написана на ній програма може бути використана тільки в цьому середовищі.

З допомогою мов низького рівня створюються дуже ефективні й компактні програми, тому що розробник одержує доступ до всіх можливостей процесора. З іншого боку, потрібно дуже добре розуміти пристрої комп'ютера, ускладнюється налагодження великих додатків, а результуюча програма не може бути перенесена на комп'ютер з іншим типом процесора. Подібні мови звичайно застосовують для написання невеликих системних додатків, драйверів пристроїв, модулів стикування з нестандартним устаткуванням, коли найважливішими вимогами стають компактність, швидкодія й можливість прямого доступу до апаратних ресурсів. У деяких областях, наприклад у машинній графіці, мовою асемблера пишуться бібліотеки, що ефективно реалізують потребуючі інтенсивних обчислень алгоритми обробки зображень.

Мови програмування високого рівня значно ближчі й зрозуміліші людині, ніж комп'ютеру. Особливості конкретних комп'ютерних архітектур в них не враховуються, тому створювані програми на рівні вихідних текстів легко переносяться на інші платформи, для яких створений транслятор цієї мови. Розробляти програми на мовах високого рівня за допомогою зрозумілих і потужних команд простіше, а помилок при створенні програм допускається набагато менше [20].

1.5. Огляд мов програмування високого рівня

Python

Python є об'єктно-орієнтованою мовою програмування загального призначення. Вона широко застосовується для створення вебдодатків, обробки даних, штучного інтелекту/машинного навчання та в інших напрямках.

Python є інтерпретованою мовою. Вона дозволяє перевіряти фрагменти коду одразу, без компіляції. Це прискорює розробку та допомагає ефективніше інтегрувати системи.

Python є кросплатформною мовою. Написані на ній програми можуть працювати в різних операційних системах і у вигляді інтерпретованих сценаріїв, і у вигляді виконуваних файлів.

Врешті-решт, мова Python проста для опанування, її підтримує велика активна спільнота, а в інтернеті є багато ресурсів для її вивчення.

JavaScript

JavaScript (JS) – провідна мова всесвітньої павутини. Двигун JS є основним інструментом усіх сучасних веббраузерів. Тому понад 95% онлайн-застосунків та вебсайтів використовують її як мову програмування фронтенду.

На JavaScript можна створювати динамічні інтерактивні інтерфейси, кросплатформне програмне забезпечення, мобільні програми та віджети, браузерні ігри та інші програми.

Водночас JS використовується у фулстек-розробці, бо виконується і в клієнті, і на сервері.

Основні переваги JS:

- швидкий та простий кодинг;
- широка функціональність;
- велика колекція бібліотек і фреймворків.

Java

Java є популярною мовою програмування, яку використовують для розробки мобільних застосунків, вебзастосунків, застосунків для робочого столу, ігор, корпоративних застосунків, і це ще не повний перелік.

Java є кросплатформною мовою. Вона виконується віртуальними машинами, які створено для різних ОС, тому той самий код, написаний на ній, працюватиме на різних платформах.

Java, як і Python, має багато бібліотек з відкритим вихідним кодом і велику спільноту, в якій можна знайти підтримку з боку досвідчених програмістів.

Java підтримує передові практики програмування. Їх вивчення стане в нагоді для використання і в інших мовах програмування.

C#

C# – це об’єктно-орієнтована мова загального призначення. Її використовують для створення вебзастосунків, програм для робочого столу, мобільних пристроїв, ігор та іншого програмного забезпечення.

Синтаксис C# нагадує C/C++ та Java/JavaScript, тому її легко опанувати тим, хто знайомий з цими мовами.

C# є компонентно-орієнтованою мовою. Її мовні конструкції підтримують цю концепцію, завдяки чому мова C# є природною мовою створення та використання програмних компонентів.

C# створено для платформи .NET, вона компілюється в код на проміжній мові, який може взаємодіяти з кодом, який написано для тієї самої платформи мовами F#, Visual Basic і C++.

На додаток до вищесказаного – платформа .NET має безліч бібліотек. Їх розподілено за просторами імен, вони і надають такі можливості, як-от читання/запис файлів, робота з рядками, аналіз XML, надають платформи для створення вебзастосунків та елементи керування Windows Forms.

C

C є процедурною імперативною мовою програмування загального призначення. Її було створено в 1972 році для розробки операційної системи UNIX. Незважаючи на свій вік, мова C широко використовується і зараз.

Серед основних можливостей мови C – низькорівневий доступ до пам’яті, простий набір ключових слів та охайний стиль. Завдяки таким можливостям C підходить для системного програмування, наприклад, розробки операційних систем та компіляторів.

C дуже швидка порівняно з іншими мовами програмування, зокрема Java та Python.

Якщо ви знаєте C, то не матимете проблем із вивченням інших популярних мов програмування, наприклад Java, JavaScript, C++, C#, Python тощо завдяки подібному синтаксису.

C++

Мову C++ розроблено як розширення C, і синтаксис цих мов майже однаковий. На відміну від C, C++ підтримує класи та об’єкти. Це одна з найпоширеніших мов у розробці ігор. Її широко використовують для написання драйверів, керування апаратним забезпеченням, навчання та досліджень. Інтерфейс Windows написано на C++.

C++ є компільованою мовою загального призначення зі статичною типізацією. У ній поєднано можливості мов високого та низького рівнів, тому її називають мовою середнього рівня. C++ підтримує процедурне, об’єктно-орієнтоване та узагальнене програмування.

Зліт її популярності у 2022 році зумовлено, серед іншого, публікацією нових мовних стандартів з цікавими можливостями. *Першою віхою* став C++ 11. Цей стандарт було опубліковано 2011 року, і він став першою значною зміною з 1998 року. *Другою віхою* – стандарт C++ 20, у якому, зокрема, запроваджено модулі.

Можливо, завдяки цьому C++ утримуватиме свої позиції в рейтингу TIOBE.

PHP

PHP є абревіатурою від PHP: Hypertext Preprocessor (PHP). За допомогою PHP можна створювати динамічні вебсайти або вебпрограми, які зберігають контент у базах даних. З ним використовуються такі бази, як-от MySQL, PostgreSQL, Informix, Microsoft SQL Server, Oracle і Sybase.

Спочатку мова PHP була невеликим проєктом із відкритим вихідним кодом. Згодом більше користувачів стали розуміти, наскільки ця мова ефективна, і тоді вона отримала розвиток.

PHP є мовою сценаріїв. Її код вбудовується в код HTML і перетворюється на код на HTML на сервері.

PHP виконується досить швидко. Наприклад, модуль для Apache скомпільовано, за рахунок чого швидкість його виконання набагато більша, ніж

під час інтерпретації коду. Запущений сервер MySQL виконує складні запити з об’ємними результатами у рекордні терміни.

R

R є мовою програмування й програмним середовищем для статистичного аналізу, створення графічних представлень і звітності. Її скомпільовані бінарні версії доступні для таких операційних систем, як Linux, Windows і Mac.

R надає розробникам безліч статистичних і графічних технологій (лінійне та нелінійне моделювання, статистичні тести, аналіз часових рядів, класифікацію, кластеризацію тощо).

Поміж інших, R має такі можливості:

- ефективна обробка й зберігання даних;
- набір операторів для обчислень на масивах, зокрема матрицях;
- велика узгоджена колекція інструментів середнього рівня для аналізу даних;
- графічні засоби для аналізу й візуалізації даних або безпосередньо на комп’ютері, або у вигляді роздруківки;
- якісно розроблена, проста й ефективна мова програмування S з умовними операторами, циклами, користувацькими рекурсивними функціями та засобами введення-виведення.

Swift

Swift створено з метою розробки для iOS та OS X. Swift поєднує в собі найкращі риси C й Objective-C.

Swift використовується для розробки програмного забезпечення для телефонів, настільних комп’ютерів та інших пристроїв, які виконують код. Це безпечна, швидка й інтерактивна мова програмування.

Swift усуває безліч помилок програмування за рахунок застосування сучасних шаблонів:

- змінні завжди ініціалізуються перед використанням;
- індекси масивів перевіряються на помилки виходу за межі діапазону;
- цілі числа перевіряються на переповнення;

- необов’язкові значення гарантують явну обробку значень nil;
- керування пам’яттю здійснюється автоматично;
- обробка помилок забезпечує контрольоване відновлення під час непрогнозованих збоїв;
- код на Swift компілюється та оптимізується, щоб повною мірою використати можливості сучасного обладнання.

Ruby

Ruby є мовою сценаріїв, яка працює на безлічі платформ, зокрема Windows, Mac OS та на різних варіантах UNIX. У Ruby поєднано риси Perl, Smalltalk, Eiffel, Ada та Lisp, а також збалансовано функціональне та імперативне програмування.

Серед можливостей Ruby:

- обробка виняткових ситуацій, як у Java та Python;
- збиральник сміття для об’єктів;
- написання розширень для Ruby на C простіше, ніж Perl або Python, Ruby можна вбудовувати в ПЗ як мову сценаріїв;
- Ruby може завантажувати розширення динамічно, якщо дозволяє ОС;
- підтримка потоків виконання не залежить від ОС, тому на будь-якій платформі, на якій виконується Ruby, можна користуватися багатопоточністю, незалежно від того, чи підтримує її ОС, навіть у MS-DOS.

TypeScript

TypeScript є надбудовою над JavaScript, яка забезпечує підтримку статичної типізації, класів та інтерфейсів. Його компілятор приймає такий код і перетворює його на JavaScript. На TypeScript написано платформу Angular 2.0.

Коли ви опануєте TypeScript, ви зможете писати програми з підтримкою ООП та компілювати їх у JavaScript і для серверної, так і для клієнтської частини.

Програмісти, знайомі з ООП, легко опанують TypeScript. А знаючи TypeScript, вони зможуть набагато швидше розробляти вебзастосунки завдяки ефективній інструментальній підтримці.

1.6. Огляд мов програмування низького рівня

Низькорівнева мова програмування – мова програмування наближена до машинного коду. Перші комп'ютери доводилось програмувати двійковими машинними кодами. Проте програмувати таким чином – доволі трудомістке і важке завдання з тої причини, що людям важко запам'ятовувати цифрові позначення команд. Для спрощення цього завдання почали з'являтися мови програмування низького рівня, які дозволяли задавати машинні команди в зрозумілішому для людини вигляді. Для перетворення їх у двійковий код були створені спеціальні програми – транслятори [19].

Мова асемблера (англ. assembly language) – мова програмування низького рівня для програмованої обчислювальної системи (мікропроцесора, мікроконтролера, комп'ютера або іншого програмованого пристрою), в якій існує сувора відповідність між операторами мови та машинними командами. Асемблер також називають символічним машинним кодом або мнемокодом.

Кожна мова асемблера специфічна для конкретної комп'ютерної архітектури. На відміну від цього, програми на мовах програмування високого рівня, як правило, здатні виконуватися на декількох архітектурах, хоча вимагають специфічної для платформи інтерпретації або компіляції.

Програма мовою асемблера перетворюється у виконуваний машинний код за допомогою програми-асемблера. Процес перетворення називають асемблюванням або збіркою (англ. assembly, assembling). У більшості випадків цей процес відбувається у два етапи: асемблювання і компонування (англ. linking).

Асемблер (англ. assembler – складальник) – комп'ютерна програма, що генерує об'єктний двійковий код з заданої (як правило у текстовому вигляді) послідовності машинних інструкцій. Кожна інструкція має свою мнемоніку, і складається з символічної назви (наприклад, MOV – від англ. move, «перемістити»), за якою опціонально можуть слідувати операнди. Асемблер також обчислює значення констант і здійснює резолвінг символічних імен (і, якщо потрібно, записує у об'єктний файл адреси, які потрібно модифікувати під

час компонування чи завантажування програми). Використання символьних посилань і автоматизація обчислень адрес є однією з ключових особливостей асемблера, яка звільняє програміста від кропітких ручних обчислень (які доводилося б робити навіть при додаванні чи вилученні однієї інструкції програми). Більшість асемблерів також мають макрокоманди, що дозволяють замінювати у програмі повторювані фрагменти коду викликом макроса.

У простому випадку асемблер переводить одну операцію (машинну команду з параметрами) початкової програми у відповідний машинний код (так звана трансляція «один в один»). При цьому взаємне розташування кодів інструкцій у об'єктному модулі визначається порядком операцій у початковій програмі і повністю залежить від програміста. Для розширення можливостей низькорівневого програмування асемблери можуть реалізовувати підтримку макрокоманд – груп команд, що можуть вставлятися до програми потрібну кількість разів. В цьому випадку перед трансляцією проводиться заміна макрокоманд макророзширеннями – послідовностями команд на базовій мові відповідно до макроозначень. У останніх задається прототип макрокоманди зі структурою списку параметрів і процедура генерування макророзширення. Транслятор, що виконує функції макрогенератора і асемблера, називається макроасемблером. При трансляції з мов високого рівня асемблер нерідко використовується для виконання завершальної фази трансляції.

Трансляція зазвичай вимагає двох переглядів початкової програми: при першому перегляді здійснюється розподіл пам'яті і надання значень символічним іменам; при другому – формується робоча програма у вигляді об'єктного файлу. В процесі трансляції асемблер проводить повний синтаксичний контроль початкової програми, забезпечуючи при цьому достатньо точну діагностику помилок за місцем і характером.

Вважається, що мова асемблера є низькорівневою (на противагу мовам програмування високого рівня, що частково чи повністю абстрагуються від деталей реалізації команд процесора). Чим нижчий рівень мови програмування, тим ближча специфіка роботи програми до самого процесора, для якого вона й

була написана. Вважається, що мови низького рівня складніші й потребують більш вузької спеціалізації програміста, оскільки програма написана на асемблері для одного типу процесорів виявиться не завжди придатною для роботи з іншими процесорами. З іншого боку, програми написані на асемблері компактні та швидкі, що теж важливо.

Команди мови асемблера відповідають машинним кодам відповідного мікропроцесора чи мікроконтролера. Фактично, мова асемблера являє собою зручнішу символьну форму запису машинних команд. Як наслідок, програми написані для одного типу процесорів, на іншому не будуть функціонувати. Мова асемблера також містить засоби для створення міток та переходів, що необхідно для створення циклів та розгалужень. Можуть бути наявні засоби для створення макросів, процедур. Кожне сімейство (модельний ряд) мікропроцесорів має свій набір команд і, відповідно, свій набір інструкцій на мові асемблера.

Основні особливості мови асемблера:

- мінімальна кількість надлишкового коду (використання меншої кількості команд та звернень в пам’ять). Як наслідок – велика швидкість і менший розмір програми;
- великі обсяги коду, велике число додаткових дрібних завдань;
- погана читабельність коду, труднощі підтримки (налагодження, додавання можливостей);
- труднощі реалізації парадигм програмування та будь-яких інших скільки-небудь складних конвенцій, складність спільної розробки;
- мала кількість доступних бібліотек, їх низька сумісність;
- безпосередній доступ до апаратури: портам введення-виведення, особливим регістрам процесора;
- максимальна «підгонка» для потрібної платформи (використання спеціальних інструкцій, технічних особливостей «заліза»);
- неможливість роботи на платформах з іншою (несумісною) архітектурою [18].

Байт-код або *байткод* (англ. byte-code), іноді також використовується термін *псевдокод* – машино-незалежний код низького рівня, що генерується транслятором і виконується інтерпретатором. Більшість інструкцій байт-коду еквівалентні одній або кільком командам Асемблера. Трансляція в байт-код займає проміжне положення між компіляцією в машинний код і інтерпретацією.

Байт-код називається так тому, що довжина кожного коду операції – один байт, але довжина коду команди різна. Кожна інструкція є однобайтовим кодом операції від 0 до 255, за яким слідують такі параметри, як регістри або адреси пам'яті. Це в типовому випадку, але специфікація байт-коду значно відрізняється в мовах програмування.

Програма на байт-коді зазвичай виконується інтерпретатором байт-коду (його ще називають віртуальною машиною, оскільки він подібний до комп'ютера). Перевага – в портативності, тобто один і той же байт-код може виконуватися на різних платформах і архітектурі – цю перевагу мають всі мови, що інтерпретуються. Проте, оскільки байт-код зазвичай є менш абстрактним, компактним і більш «комп'ютерним» ніж початковий код, ефективність байт-коду зазвичай вища, ніж чиста інтерпретація початкового коду, призначеного для правки людиною. З цієї причини багато сучасних інтерпретованих мов насправді транслюють в байт-код і запускають інтерпретатор байт-коду. До таких мов відносяться Perl, PHP і Python. Програми на Java зазвичай передаються на цільову машину у вигляді байт-коду, який перед виконання транслюється в машинний код «на льоту» – за допомогою JIT-компіляції. У стандарті відкритих завантажувачів Open Firmware фірми Sun Microsystems байт-код представляє оператори мови Forth.

В той же час можливе створення процесорів, для яких даний байт-код є безпосередньо машинним кодом (такі процесори існують, наприклад, для Java і Forth).

Також деякий інтерес представляє p-код (p-code), який схожий на байт-код, але фізично може бути менш лаконічним і сильно варіюватися по довжині інструкції. Він працює на дуже високому рівні, наприклад «надрукувати рядок»

або «очистити екран». Р-код використовується в деяких реалізаціях BASIC і Паскаля [28].

1.7. Етапи розв'язування задач за допомогою персонального комп'ютера

Варто виділити такі етапи підготовки та розв'язання завдань за допомогою персонального комп'ютера (ПК): постановка завдання, вибір методу, розробка алгоритму, складання програми, введення її в пам'ять ПК, налагодження програми, її тестування та підготовка документації. Не можна ігнорувати жодного з цих етапів.

Рішення будь-якого завдання на ЕОМ складається з декількох етапів, серед яких варто виділити основні:

1. Постановка завдання;
2. Формалізація (математична постановка задачі);
3. Вибір (або розробка) методу рішення;
4. Розробка алгоритму (алгоритмізація);
5. Складання програми (програмування), введення її в пам'ять ПК;
6. Налагодження програми, її тестування;
7. Обчислення й обробка результатів.

Послідовне виконання вказаних етапів становить повний цикл розробки, налагодження й обчислення програми. Наведений розподіл є умовним, але не варто ігнорувати жодного з цих етапів. Розглянемо найбільш загальні й необхідні етапи. Разом із зазначеними користувач ПК у процесі рішення завдання може виконувати також такі етапи, як вибір мови програмування, опис структури даних, оптимізація програми, тестування, документування й ін.

Постановка завдання. При постановці завдання першорядну увагу треба приділити з'ясуванню кінцевої мети й виробленню загального підходу до досліджуваної проблеми; з'ясуванню, чи існує рішення поставленого завдання й чи єдине воно; вивченню загальних властивостей розглянутого фізичного явища або об'єкта, аналізу можливостей конкретного ПК і даної системи програмування. На цьому етапі потрібне глибоке розуміння сенсу поставленого

завдання. Правильно сформулювати завдання іноді не менш складно, ніж його вирішити.

Формалізація. Формалізація, як правило, полягає у побудові математичної моделі розглянутого явища, коли в результаті аналізу сутності завдання визначаються обсяг і специфіка вихідних даних, вводиться система умовних позначок, встановлюється приналежність розв'язуваного завдання до одного з відомих класів завдань і вибирається відповідний математичний апарат. При цьому потрібно вміти сформулювати мовою математики конкретні завдання фізики, механіки, економіки, технології й т. п. Для успішного подолання цього етапу потрібні не тільки солідні відомості з відповідної предметної галузі, але й гарне знання обчислювальної математики, тобто тих методів, які можуть бути використані при розв'язанні завдання на комп'ютері.

Повна постановка багатьох складних завдань нездійсненна засобами обчислювальної техніки. Тому ці завдання потрібно спрощувати. Грамотне спрощення завдання неможливе без гарного подання про те, які фактори й параметри найбільш важливі для досліджуваного завдання, а які - менш істотні. При цьому дуже важливо знати, яка з можливих розрахункових схем може привести до спрощення обчислювального характеру, обумовлених вибором обчислювального методу. Якщо наявних засобів недостатньо, тоді необхідно розробити новий підхід, нові методи дослідження.

Вибір методу рішення. Після того як визначено математичне формулювання завдання, треба вибрати метод його рішення. Загалом, застосування будь-якого методу приводить до побудови ряду формул і формулювання правил, що визначають зв'язки між цими формулами. Все це розбивається на окремі дії так, щоб обчислювальний процес міг бути виконаний машиною. При виборі методу треба враховувати, по-перше, складність формул і співвідношень, пов'язаних з тим або іншим чисельним методом, по-друге, необхідну точність обчислень і характеристики самого методу. На вибір методу рішення великий вплив мають смаки й знання самого користувача.

Цей етап – найважливіший у процесі рішення задачі. З ним зв’язані численні невдачі, які є результатом легковажного підходу до помилок обчислень. При розв’язуванні задачі на ПК необхідно пам’ятати, що будь-який одержуваний результат є наближеним! Якщо відомо алгоритм точного рішення, то крім випадкових помилок (збоїв у роботі ПК), можливі помилки, пов’язані з обмеженою точністю подання чисел у ПК. При обчисленнях, що полягають у знаходженні результату із заданим ступенем точності, виникає додаткова погрішність, яку, якщо можливо, оцінюють на даному етапі (до виходу безпосередньо на ПК). Ця погрішність визначається обраним чисельним методом рішення завдання.

Розробка алгоритму. Даний етап полягає в розкладанні обчислювального процесу на можливі складові частини, установленні порядку їхнього проходження, описі змісту кожної такої частини в тій або іншій формі й наступній перевірці, що повинна показати, чи забезпечується реалізація обраного методу. У більшості випадків не вдається відразу одержати задовільний результат, тому складання алгоритму проводиться методом «спроб і усунення помилок» і для одержання остаточного варіанту потрібно кілька кроків корекції й аналізу.

Як правило, у процесі розробки алгоритм проходить кілька етапів деталізації. Спочатку складається укрупнена схема алгоритму, у якій відбиваються найбільш важливі й істотні зв’язки між досліджуваними процесами (або частинами процесу). На наступних етапах розкриваються (деталізуються) виділені на попередніх етапах частини обчислювального процесу, що мають деяке самостійне значення. Крім того, на кожному етапі деталізації виконується багаторазова перевірка й виправлення (відпрацьовування) схеми алгоритму. Подібний підхід дозволяє уникнути можливих помилкових рішень.

Орієнтуючись на великоблочну структуру алгоритму, можна швидше й простіше розробити кілька різних його варіантів, провести їхній аналіз, оцінку та обрати найкращий (оптимальний).

Ефект поетапної деталізації алгоритму багато в чому залежить від того, як здійснюється його структуризація: розчленовування алгоритмічного процесу на складові частини, що повинне визначатися не сваволею користувача (програміста), а внутрішньою логікою самого процесу. Кожний елемент великоблочної схеми алгоритму повинен бути максимально самостійним і логічно завершеним у такому ступені, щоб подальшу його деталізацію можна було виконувати незалежно від деталізації інших елементів. Це спрощує процес проєктування алгоритму й дозволяє здійснювати його розробку вроздріб одночасно кількома виконавцями.

У процесі розробки алгоритму можуть використовуватися різні способи його опису, що відрізняються за простотою, наочністю, компактністю, ступенем формалізації, орієнтацією на машинну реалізацію й іншими показниками. У практиці програмування найбільшого поширення набули:

1. Словесний запис алгоритмів;
2. Схеми алгоритмів;
3. Псевдокод (формальні алгоритмічні мови);
4. Структурограми (діаграми Нассі - Шнейдермана).

Розробка алгоритмів є в значній мірі творчим, евристичним процесом і, як правило, вимагає великої ерудиції, винахідливості, нестандартних і нетрадиційних підходів до рішення завдання.

Складання програми. Подання алгоритму у формі, що допускає введення в машину, переклад на машинну мову є завданнями етапу складання програми (програмування). Тобто розроблений алгоритм завдання необхідно викласти мовою, що буде зрозуміла комп’ютеру безпосередньо або після попереднього машинного перекладу. Від вибору мови програмування залежить процес налагодження програми, під час якого програма набуває остаточного робочого вигляду.

Налагодження програми. Складання програми являє собою трудомісткий процес, що вимагає від виконавця напруженої уваги. Практика показує, що в обчисленнях варто уникати поспішності й дотримуватися золотого правила:

«краще менше, та краще». Але на попередніх етапах стільки можливостей припуститися помилки, і як би ми ретельно не діяли, спочатку складена програма звичайно містить помилки. Машина або не може дати відповіді, або наводить неправильне рішення.

Налагодження починається з того, що програма, акуратно записана на бланку, перевіряється безпосередньо особою, що здійснила підготовку й програмування завдання. З'ясовується правильність написання програми, виявляються змістовні й синтаксичні помилки й т.п. Потім програма вводиться у пам'ять ПК і помилки, що залишилися непоміченими, виявляються вже безпосередньо за допомогою машини.

Досвідчений користувач ПК знає, що необхідний діючий контроль над процесом обчислень, який дозволяє вчасно виявляти й запобігати помилки. Для цього використовуються різного роду інтуїтивні міркування, правдоподібні міркування й контрольні формули. Користувач – початківець часто вважає налагодження зайвим, а одержання контрольних точок – неприємною додатковою роботою. Однак дуже скоро він переконується, що пошук пропущеної помилки вимагає значно більшого часу, ніж час, витрачений на контроль.

Гарантією правильності рішення, наприклад, може служити:

1. Перевірка виконання умов завдання (наприклад, для алгебраїчного рівняння знайдені корені підставляються у вихідне рівняння й перевіряються розходження лівої й правої частин);

2. Якісний аналіз завдання;

3. Перерахування (по можливості іншим методом).

Для деяких складних за структурою програм процес налагодження може зажадати значно більше машинного часу, ніж сам розв'язок на ПК, тому що погано сплановані процеси алгоритмізації, програмування і налагодження приводять до помилок, які можуть бути виявлені лише після багаторазових перевірок.

Обчислення й обробка результатів. Тільки після того як з’явиться повна впевненість, що програма забезпечує одержання правильних результатів, можна приступати безпосередньо до розрахунків по програмі. Після завершення розрахунків наступає етап використання результатів обчислень у практичній діяльності або, як говорять, етап впровадження результатів. Інтерпретація результатів обчислень знову належить до тієї предметної галузі знань, звідки виникло завдання [20].

Контрольні запитання

1. Яку мову називають мовою програмування?
2. Які мови називають машинно-орієнтованими?
3. Які мови називають мовами програмування високого рівня?
4. Як мови програмування класифікують за орієнтацією на клас задач?
5. Як мови програмування класифікують за принципами програмування?
6. Що таке транслятор?
7. Чим відрізняється компіляція від інтерпретації?
8. Поясніть терміни «мова низького рівня» й «мова високого рівня», чим вони відрізняються?.
9. Які покоління мов програмування Вам відомі?
10. Які мови програмування активно використовуються сьогодні?
11. Укажіть основні компоненти, які необхідні для створення програми.
12. Що таке інтегроване середовище програмування?
13. Охарактеризуйте основні напрямки розвитку мов програмування.
14. Що таке алгоритм? У чому полягає суть побудови алгоритмів?
15. Що таке програма?

Розділ 2. ЗНАЙОМСТВО ЗІ SCRATCH

2.1. Загальні відомості

Scratch – це інтерпретована динамічна візуальна мова та середовище програмування, що дозволяє створювати анімовані інтерактивні історії, ігри та моделі.

Scratch розроблено для дітей від 8 до 16 років, але використовується людьми різного віку. Середовище є повністю безкоштовним і доступне більш ніж 70 мовами, у тому числі і українською [38].

Scratch існує в онлайн та офлайн-версіях. Онлайн Scratch доступний за посиланням https://scratch.mit.edu/projects/editor/?tutorial=getStarted. При реєстрації в онлайн-версії користувачу доступне збереження проєктів та можливість поділитись ними на офіційному сайті. Якщо акаунт не створений, то проєкти можна зберігати на персональний комп'ютер. Завантажити офлайн-версію можна за посиланням https://scratch.mit.edu/download. Автономний редактор Scratch доступний для таких операційних систем, як Windows, macOS, ChromeOS та Android.

Окрім Scratch, для найменших доступна програма *ScratchJr*, розрахована на дітей віком 5-7 років. Відмінність ScratchJr полягає у значно простіших командах виконавця та інтерфейсі.

Особливість Scratch полягає у тому, що програма створюється шляхом поєднання блоків з командами, які керують спрайтами.

Спрайт (образ, персонаж) – це об'єкт, який виконує дії у проєкті. Образ за замовчуванням – Рудий Кіт. У бібліотеці середовища доступні різні образи, які користувач може додати до свого проєкту. Також можна додати образ з ПК або намалювати його [27].

2.2. Інтерфейс середовища програмування Scratch

Інтерфейс середовища Scratch є досить простим та інтуїтивно зрозумілим, а також доступний українською мовою. Усе, що необхідно виконувати користувачу – це перетягувати блоки в область сценарію та з'єднувати їх між собою у певній послідовності.

Вікно програмного засобу Scratch 3.2 (рис. 2.2.1) складається з таких основних елементів:

- Меню (1);
- Палітра блоків (2);
- Область сценаріїв (3);
- Сцена (4);
- Панель спрайтів (5).

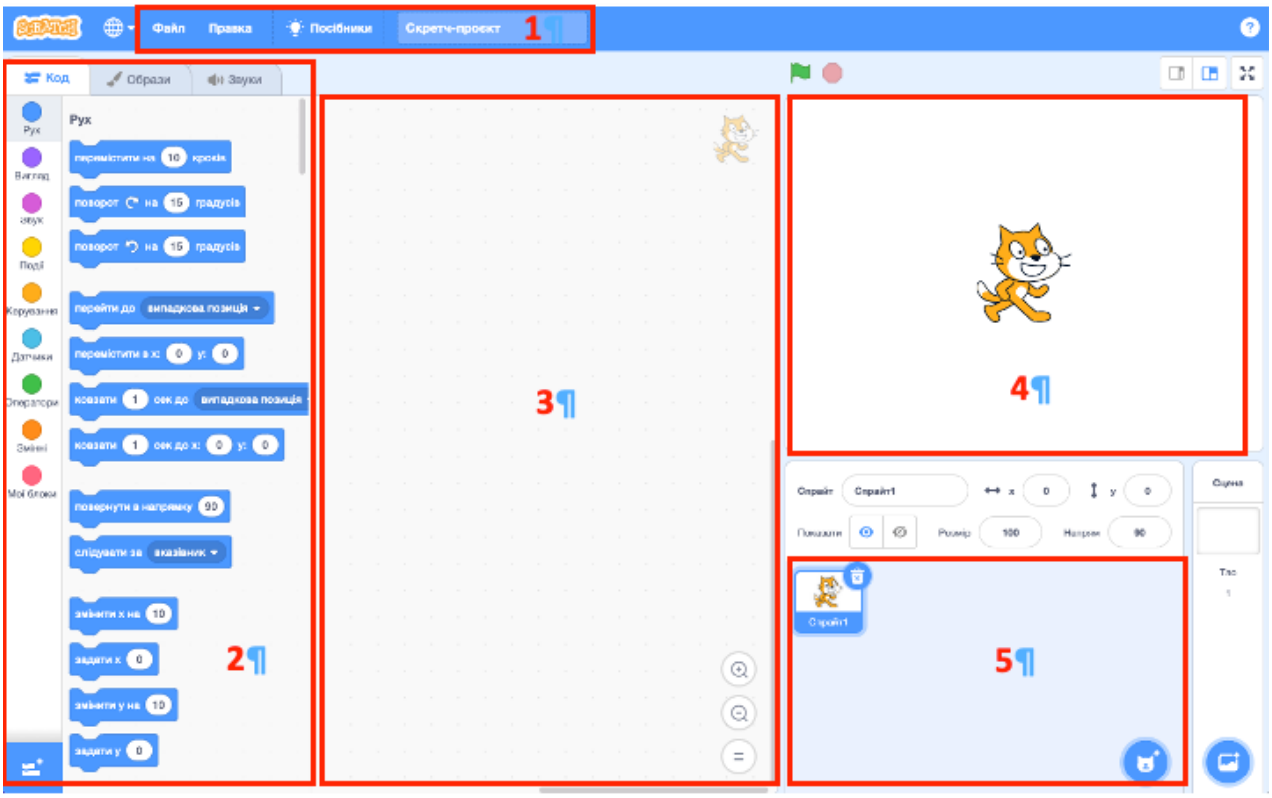


Рисунок 2.2.1. Вікно програми Scratch

Сцена призначена для розміщення спрайтів, на ній також буде видно як працюватиме готовий проєкт. На сцені видно як переміщатимуться спрайти, на

ній користувач може малювати та змінювати фон. За замовчуванням на сцені розміщено всього один спрайт – Рудий кіт.

Усі спрайти проєкту знаходяться на панелі спрайтів, що розміщена під сценою.

У центрі вікна знаходиться велика область сценаріїв, там з кольорових блоків, що розміщені на палітрі блоків, збирається сценарій.

2.3. Об'єкти. Властивості об'єктів

Програмний об'єкт – це об'єкт, що є складовою певної комп'ютерної програми, наприклад, кнопка, прапорець, вікно.

Програмними об'єктами в Scratch є *спрайти* (виконавці) та *сцена*. Спрайти мають такі властивості:

- ім'я;
- положення на Сцені;
- розмір;
- напрямок, в якому вони будуть рухатися;
- колір костюма.

Кожна властивість має значення. Переглянути та змінити значення властивостей спрайтів можна в розділі *Інформація* (рис. 2.3.1) або у вбудованому графічному редакторі.

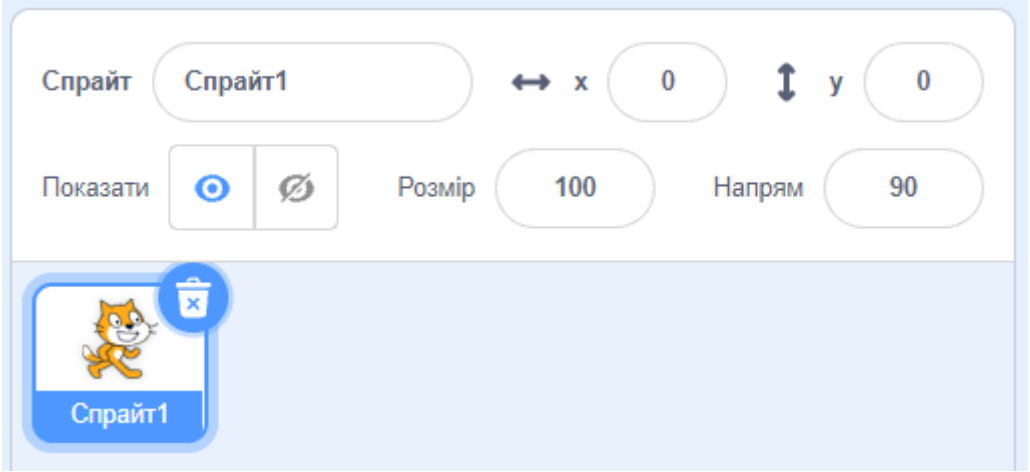


Рисунок 2.3.1. Інформація про спрайт

Створити новий спрайт у проєкті можна кількома способами, вибравши відповідну кнопку зі списку *Обрати спрайт* (рис. 2.3.2):

- обрати готовий об’єкт з бібліотеки спрайтів;
- намалювати у графічному редакторі, убудованому в середовище Scratch;
- вивантажити з файла;
- обрати спрайт випадковим чином;
- сфотографувати камерою, підключеною до комп’ютера.

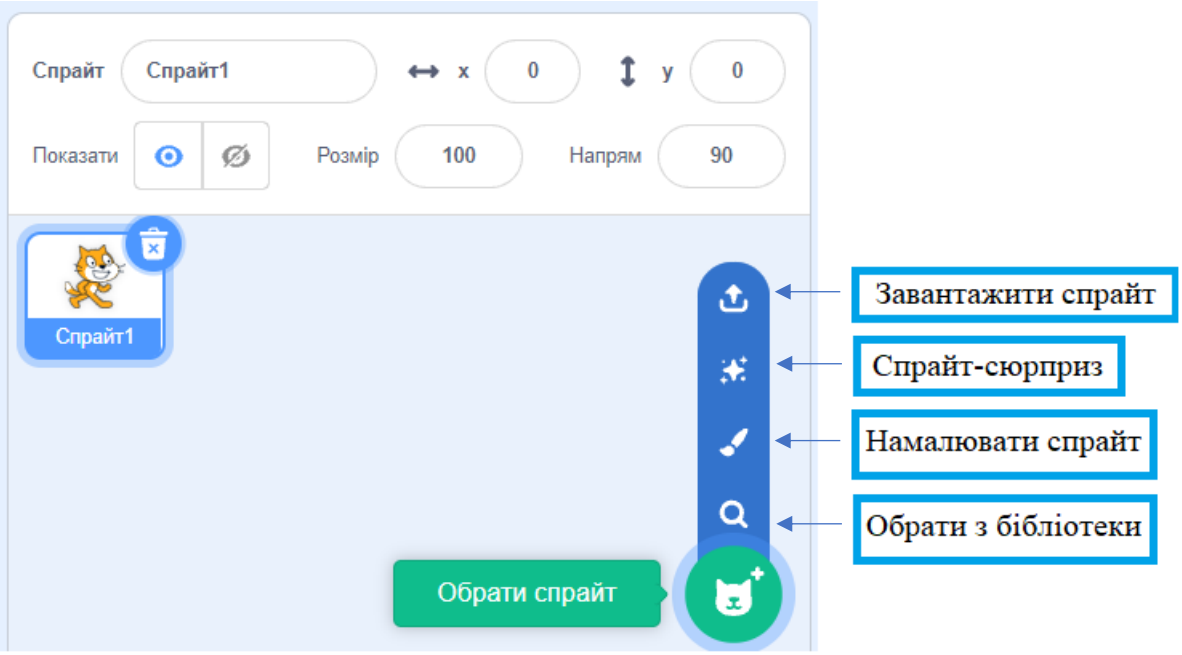


Рисунок 2.3.2. Інструмент «Обрати спрайт»

Після цього новий об’єкт з’являється на вкладці *Спрайти* і його можна використовувати у проєкті.

Сцена має такі властивості:

• розмір (480 на 360 кроків виконавця);

• тло.

На сцену накладена координатна сітка. Початок координат – центр сцени (рис. 2.3.3).

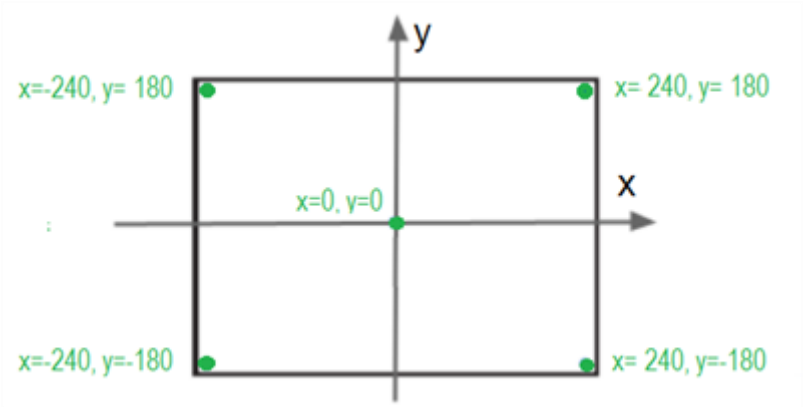


Рисунок 2.3.3. Графічне представлення сцени у системі координат

На вкладці *Тло* можна змінити зображення на тлі *Сцени*, використавши кнопки зі списку *Обрати тло* (рис. 2.3.4).

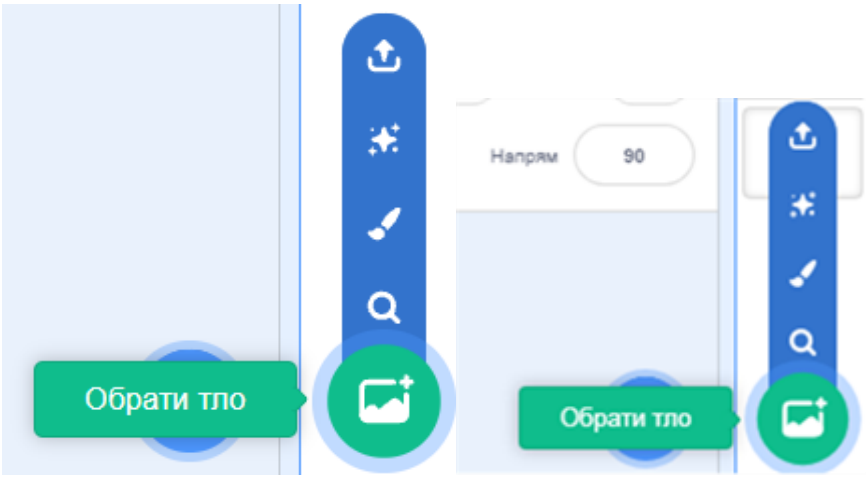


Рисунок 2.3.4. Інструмент «Обрати тло»

Змінити значення деяких властивостей спрайтів можна і під час виконання проєкту. Наприклад, через кілька кроків руху змінити колір об’єкта або його розміри, змінити положення об’єкта або взагалі сховати під час виконання деякої умови, змінити тло сцени під час переходу на новий кадр сценарію тощо.

Команди зміни властивостей спрайтів розміщені в групах *Рух* і *Вигляд* вкладки *Скрипти*.

Контрольні запитання

1. Як називається виконавець у середовищі Scratch?
2. Назвіть основні елементи вікна програми Scratch.
3. Як дізнатись поточні координати персонажа на сцені?
4. Як змінити розмір персонажа?
5. Які є способи створення нового спрайту?

Практичні завдання

Створіть гру «Космічні баталії».

Опис механіки гри:

Потрібно створити гру, в якій головним героєм є космічний шатл, який бореться з ворогами, іншими шатлами, які полюють за нашим героєм.

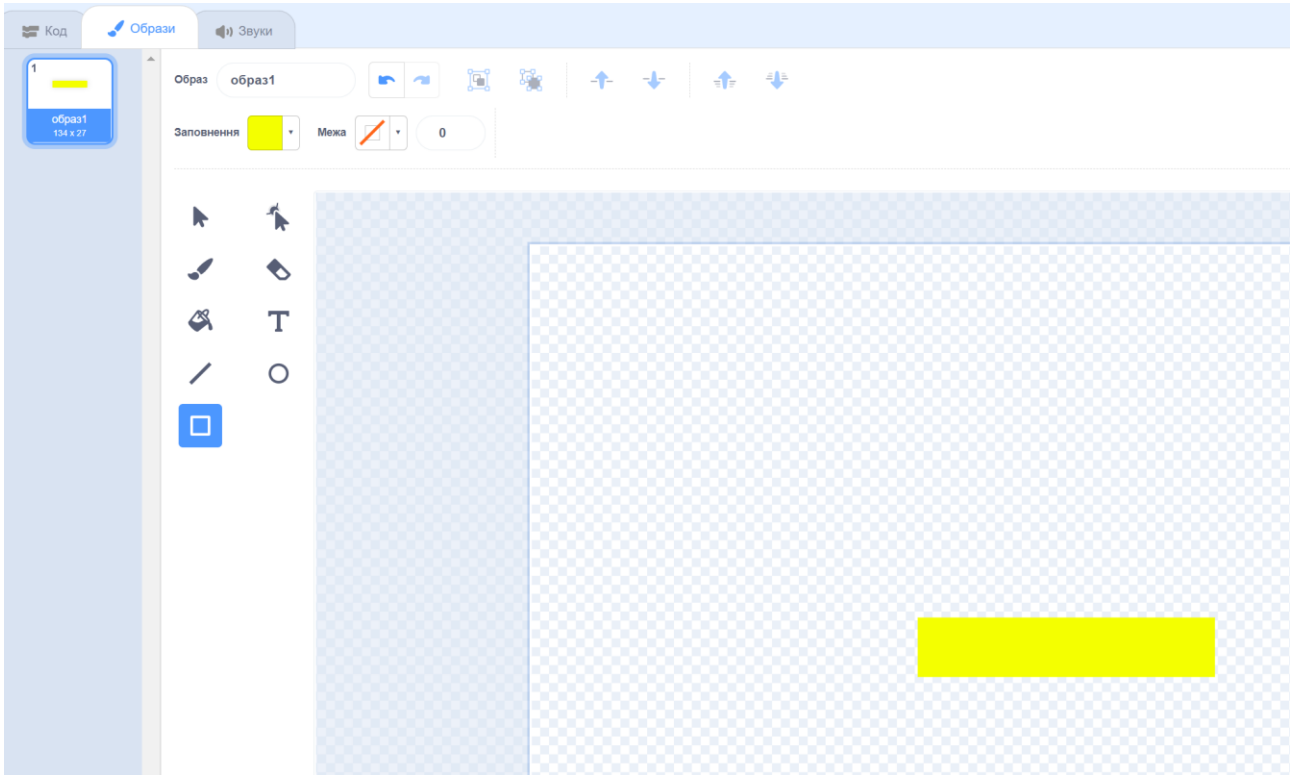
Головний герой має три життя, та може стріляти лазером у ворогів для того щоб їх знищити. Якщо ворог торкнеться головного героя, то в головного героя – 1 життя. Після втрати всіх життів, на головному екрані має з’являтись напис «Кінець гри».

Всі спрайти обирати з бібліотеки спрайтів або ж шукати у мережі Інтернет.

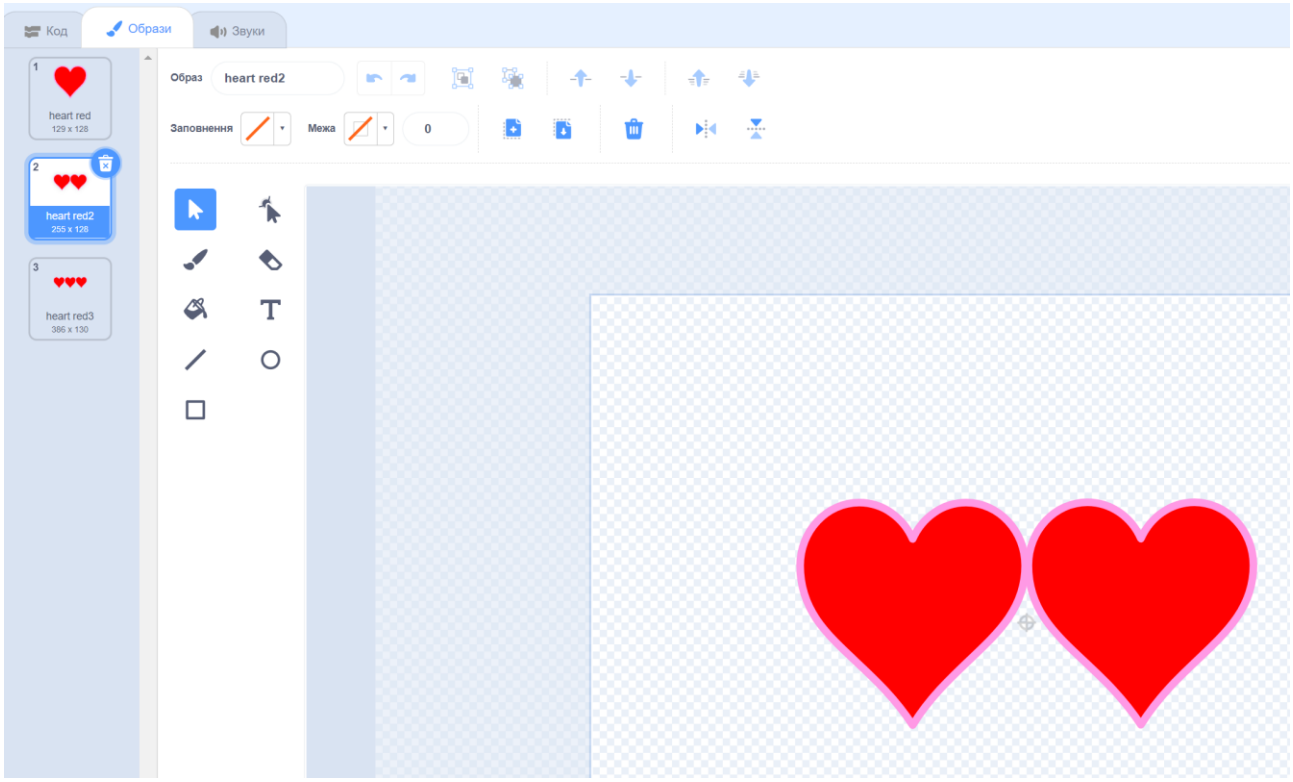
Етапи створення гри:

Елементи проєкту. Створіть проєкт, завантажте 2 спрайти, спрайт Ворога та спрайт Героя, а також завантажете спрайт тла гри.

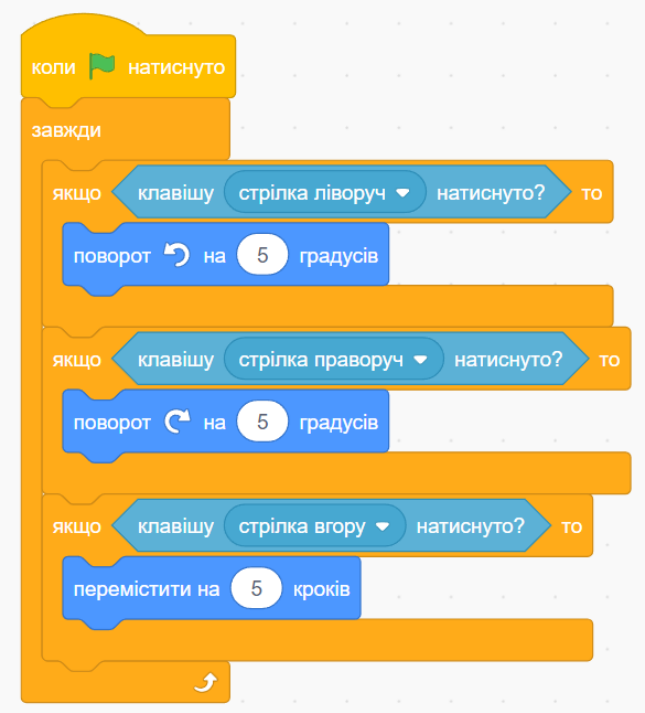
Спрайт Лазер. Намалюйте об’єкт у вигляді прямокутника, який буде виконувати роль лазера, яким стрілятиме наш Герой.



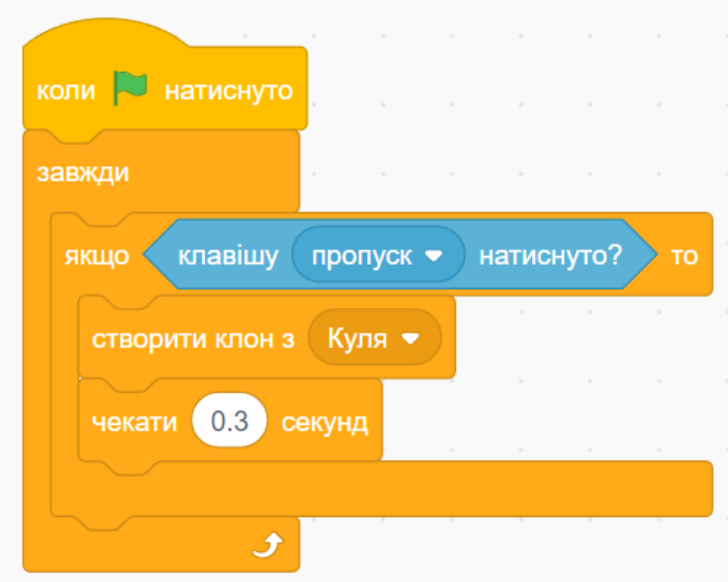
Спрайт життя. Створюємо спрайт «життя» і робимо для нього кілька костюмів.



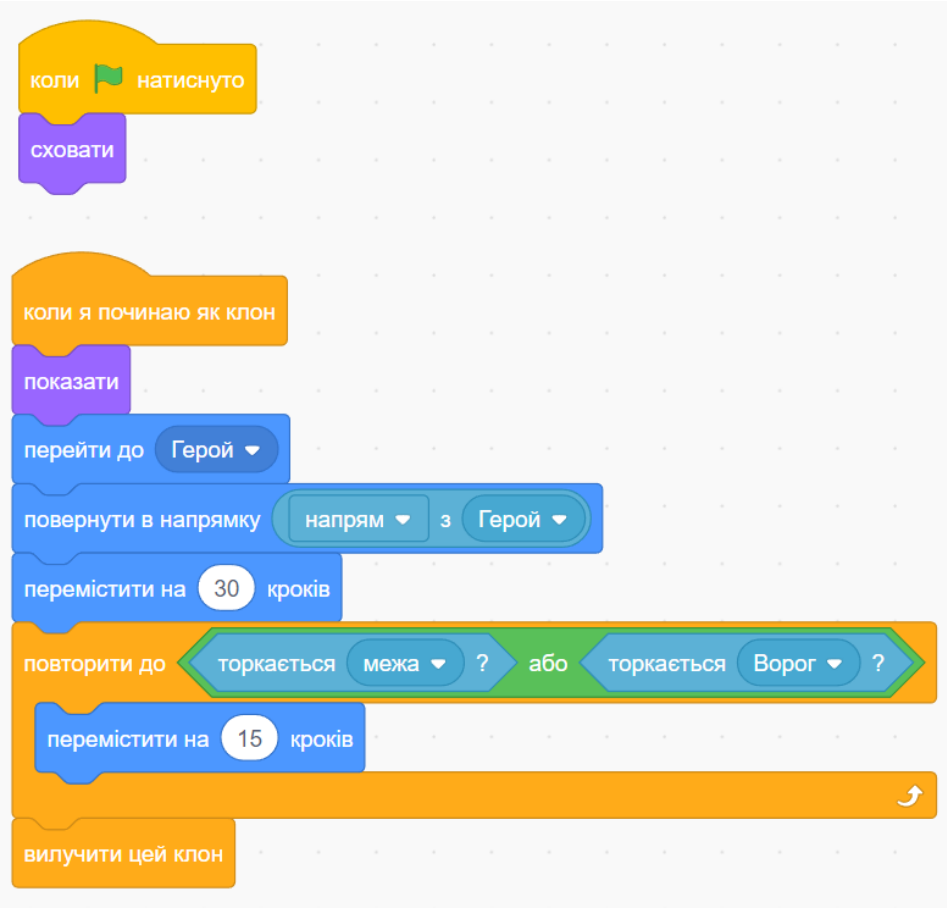
Механіка головного героя. Налаштовуємо рух головному герою з допомогою стрілок на клавіатурі.



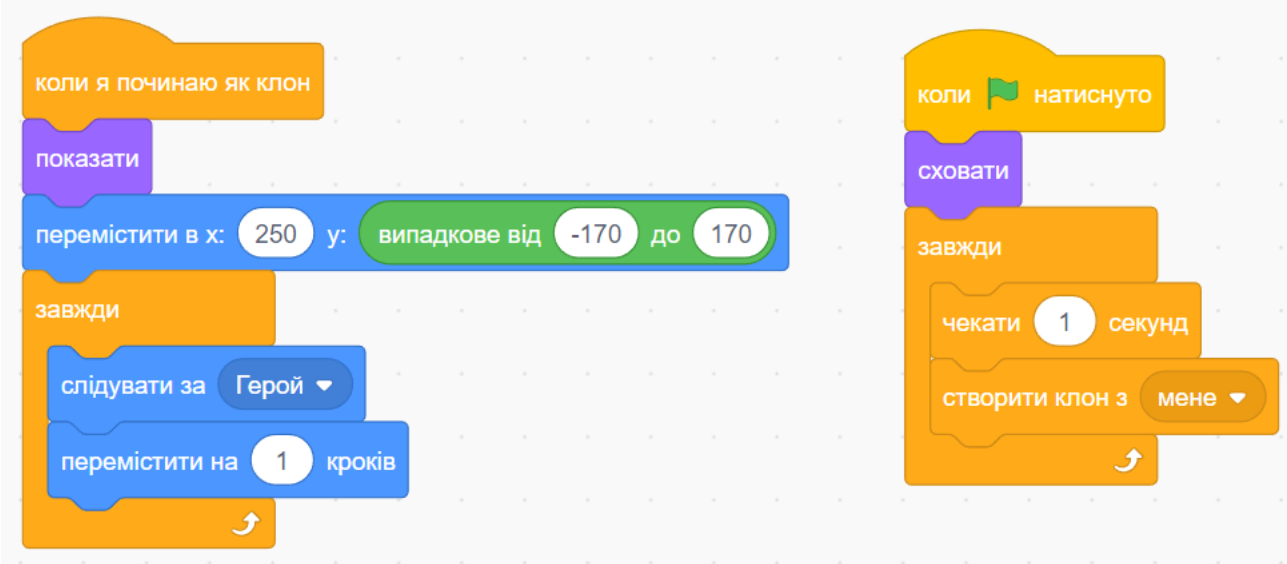
Механіка світу. Запрограмуємо ефект вистрілів лазером нашому герою.



Програмуємо спрайт **Лазера.**

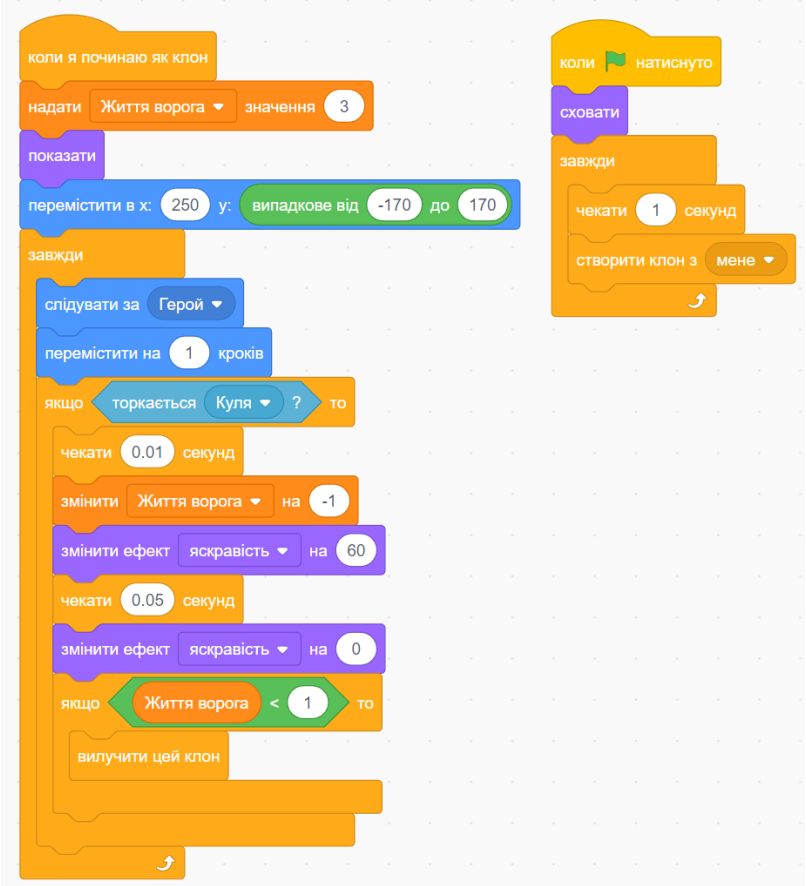


Механіка ворогів. Вороги мають з’являтися з правого краю сцени і рухаються до героя, клонуючи себе. Для цього створюємо наступний код:

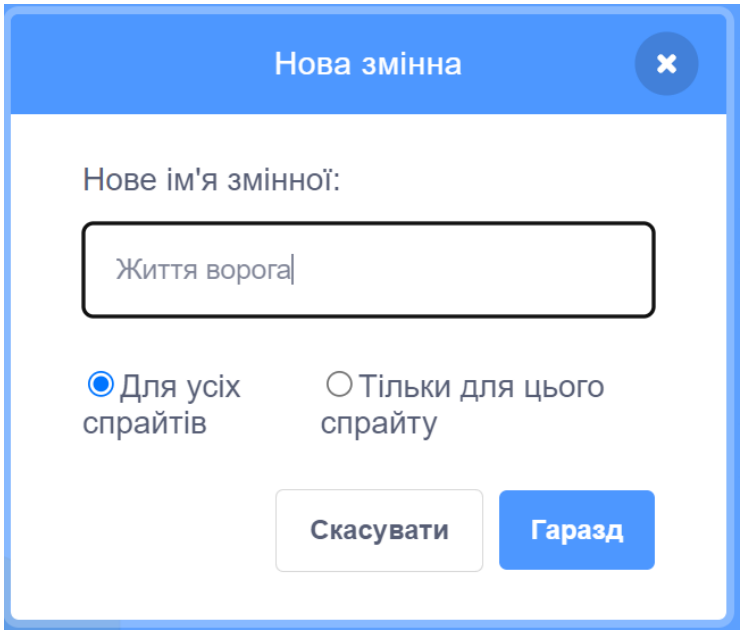


При попаданні лазера у ворога він починає мерехтіти. Для цього модернізуємо блоки ворога

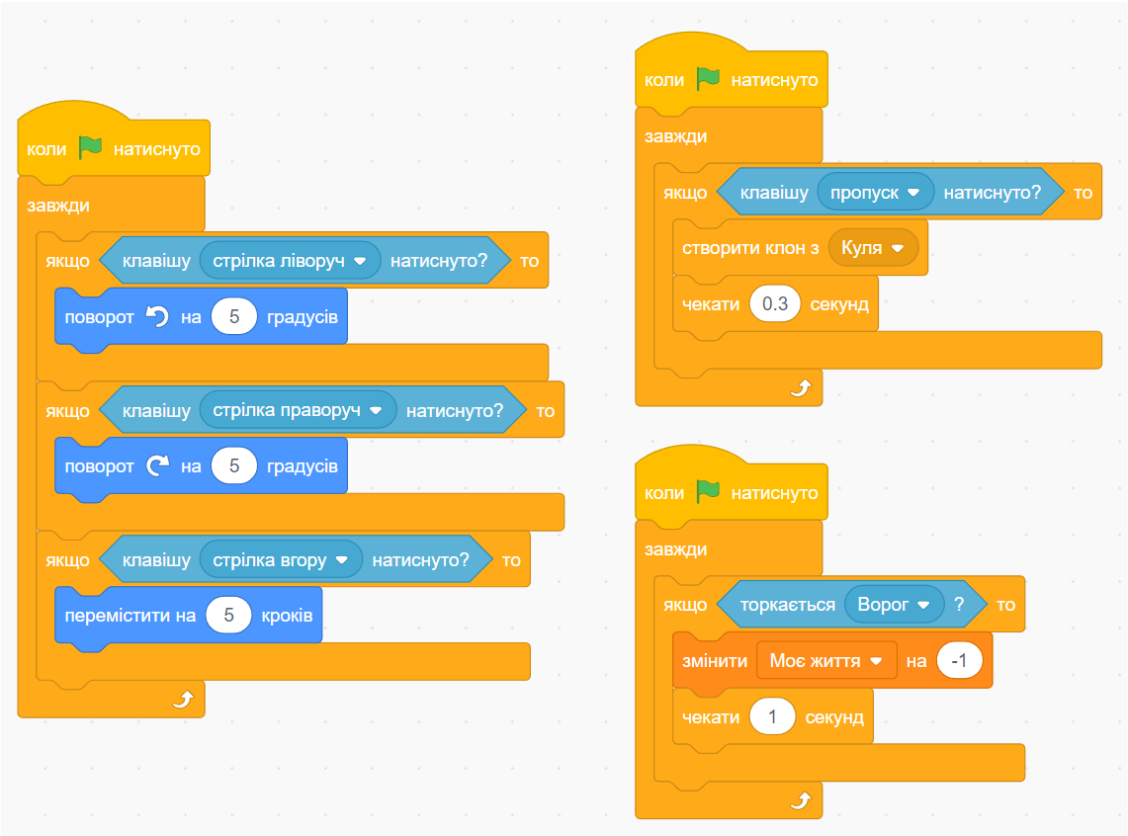
Створюємо для ворогів *змінну* «Життя ворога» і доповнюємо скрипт.



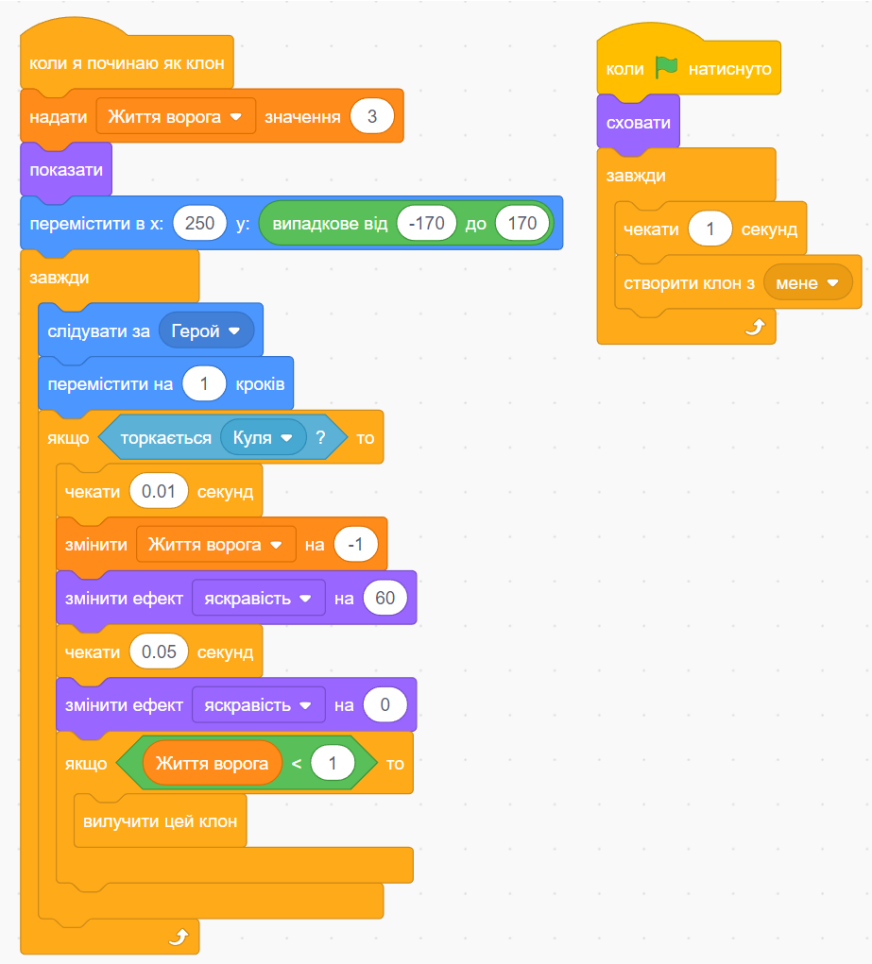
Змінна «життя». Створіть змінну для всіх спрайтів і назвіть її «Моє життя». З її допомогою ми будемо міняти можливу кількість влучень.



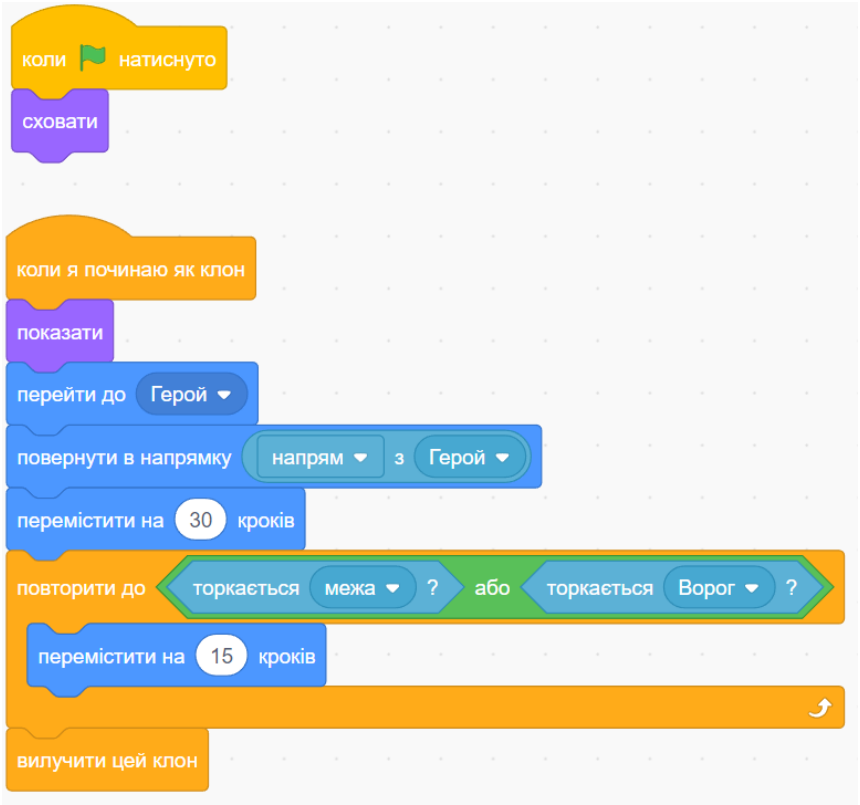
Фінал. Підсумковий скрипт героя має мати такий вигляд:



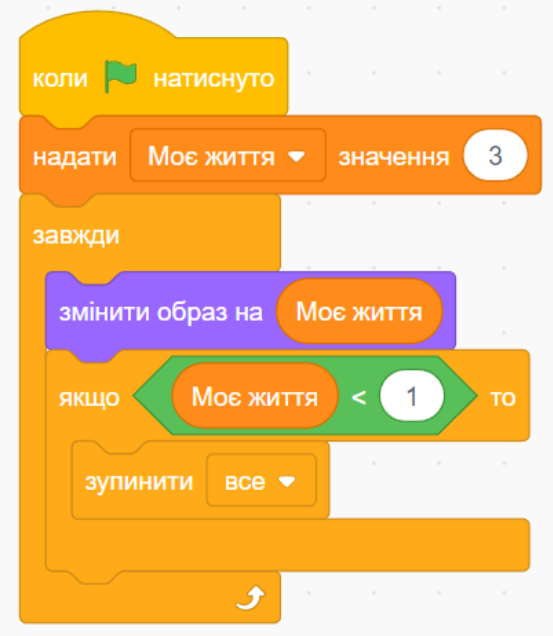
Підсумковий скрипт ворога має мати вигляд:



Підсумковий скрипт спрайта **Лазер:**



Підсумковий скрипт спрайта **Життя:**



Завдання для самостійної реалізації:

1. Домалювати додатковий варіант фону з написом «game over», налаштувати момент його появи.

2. Придумати додаткових ворогів, боса, що з’явиться в певний час або при виконанні конкретних умов

3. Зробити систему набору очок, додавши ще одну змінну і визначивши момент додавання балів, а також їх кількість

Створіть гру «Динозавр та перешкоди».

Опис механіки гри:

Потрібно створити гру, в якій головним героєм є динозавр, який минає через перешкоди.

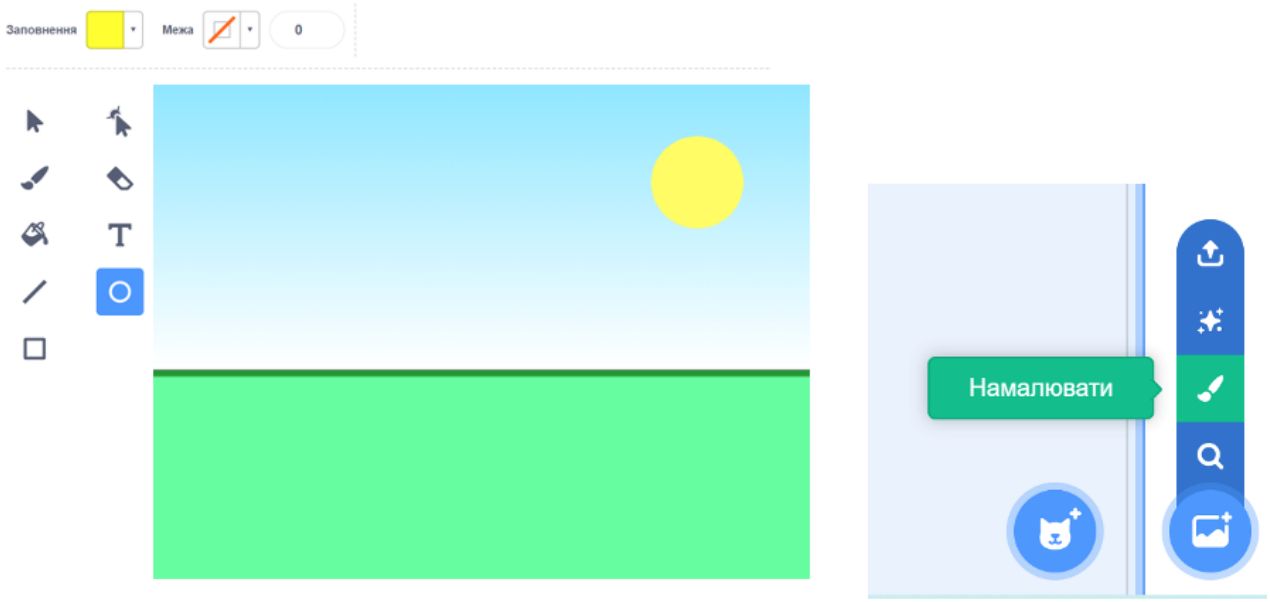
Створіть прототип браузерної гри «Динозавр», який буде минати різного типу перешкоди та набирати бали перемоги.

Всі спрайти обирати з бібліотеки спрайтів або ж шукати у мережі Інтернет.

Етапи створення гри:

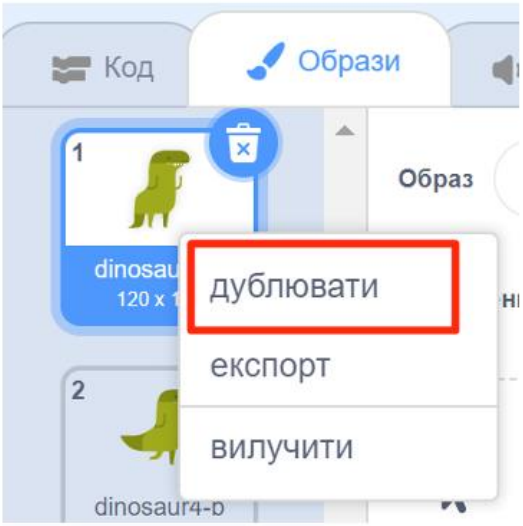
Елементи проєкту.

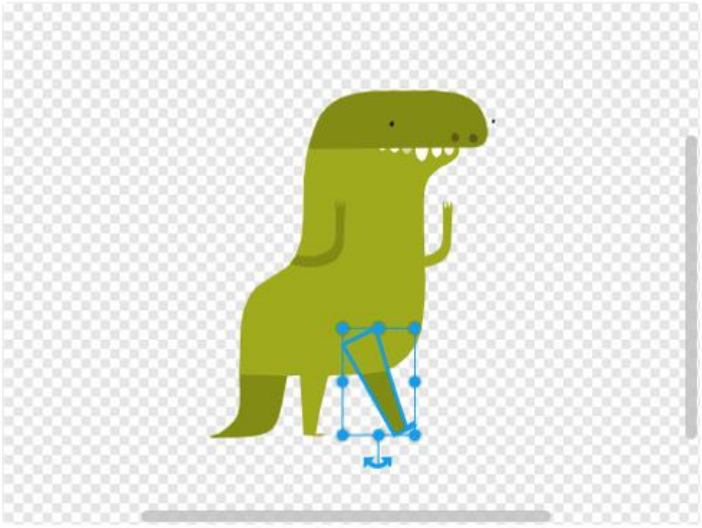
Створення фону. Створіть проєкт і самостійно намалюйте фон.



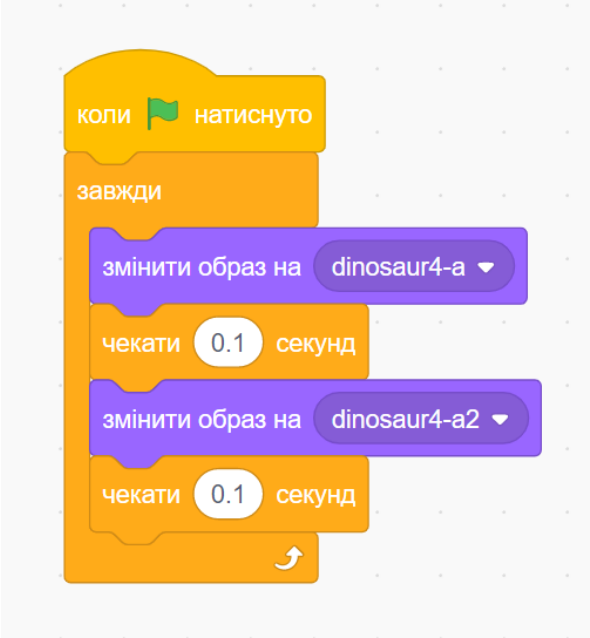
Спрайт «dinosaur4». Знайдіть в бібліотеці динозаврика «dinosaur4» і додайте в проєкт.

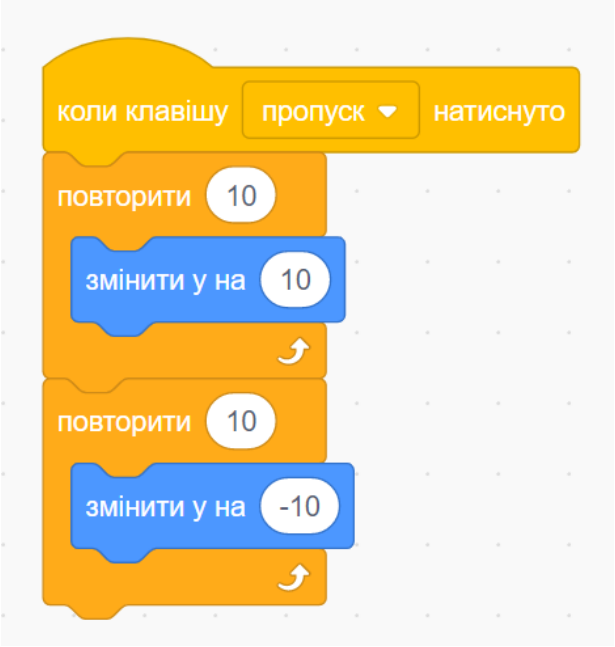
Редагування костюма. У динозавра відсутній костюм крокуючого динозавра. Перейдіть в режим редагування костюма. Створіть дублікат костюма «dinosaur4-a» поверніть передню ногу на довільний великий кут.



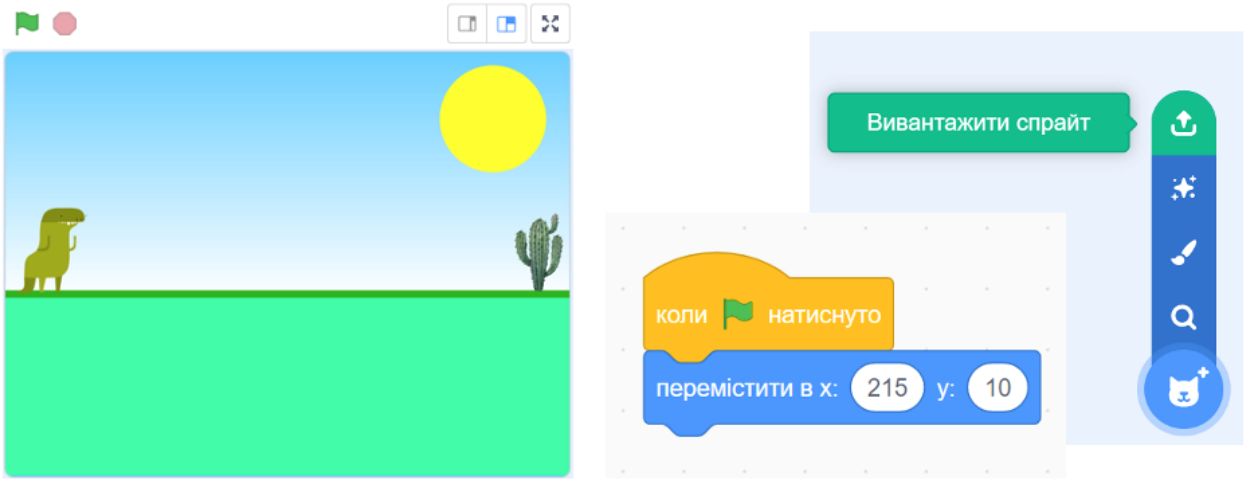


Створення анімації. Навчіть динозавра стрибати після натискання клавіші «пробіл».





Завантаження спрайта з комп'ютера. Знайдіть в Інтернеті та завантажте зображення кактуса чи іншого об'єкта, який буде перешкодою для динозавра. Зменшіть та встановіть на лінію горизонту праворуч.



Налаштування анімації.

Щоб навчити динозавтра бігти є 2 варіанта реалізації гри.

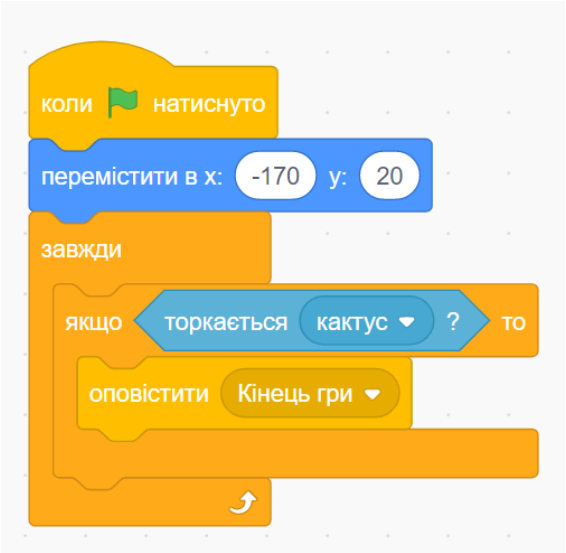
1. Запрограмувати динозавра на рух по осі X праворуч;

2. Запрограмувати перешкоду на рух по осі X ліворуч.

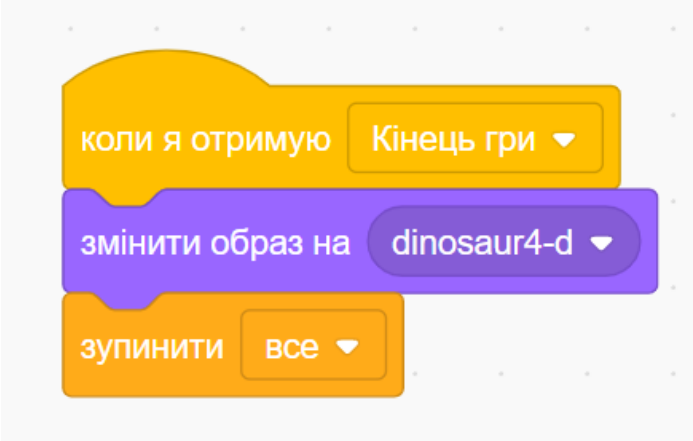
Ми будемо користуватись 2 варіантом, динозавр буде стояти на місці, а перешкоди будуть рухатись на нього, це буде створювати враження руху динозавра вперед.

Перешкода рухається ліворуч, отже координата X буде зі знаком «мінус». Щоб запрограмувати перешкоду на рух вперед використовуємо такі блоки.

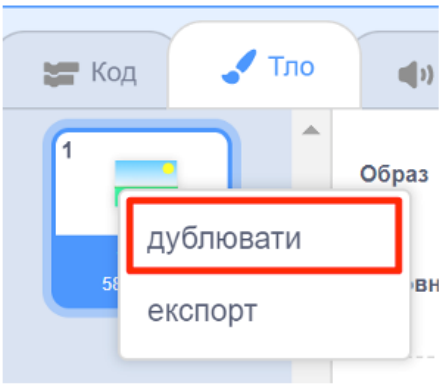
Зіткнення з перешкодою. Створіть повідомлення «кінець гри» і додайте в скрипт до динозавра. Коли динозавр торкнеться кактуса, всі об’єкти отримають повідомлення «кінець гри».



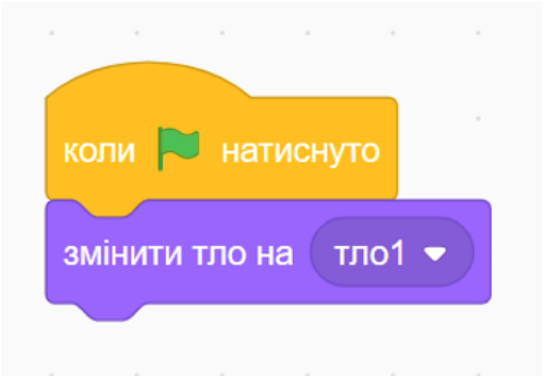
Додайте ще один скрипт динозавру:

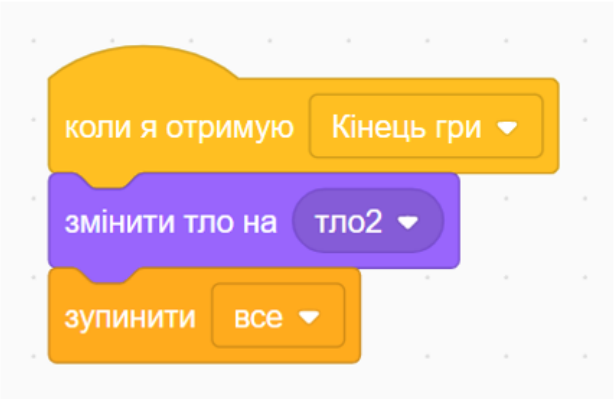


Повідомлення «кінець гри». Перейдіть на закладку «тло», створіть дублікат фону. Напишіть «game over» використовуючи інструмент «текст».

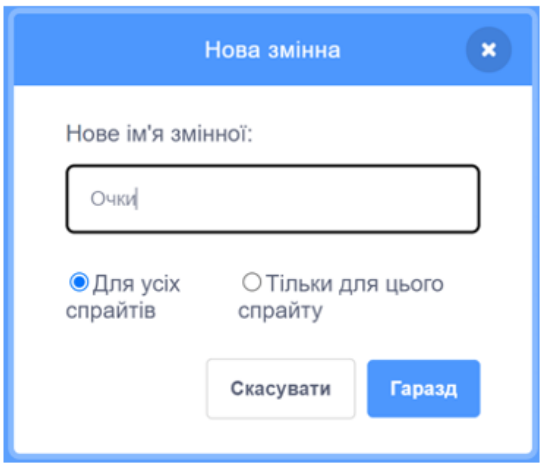


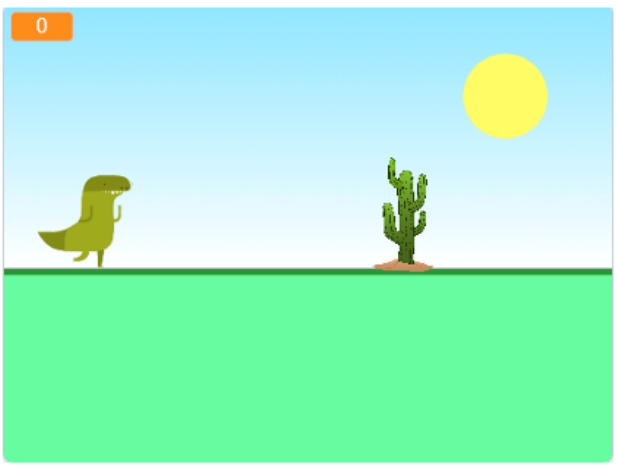
Скрипт для фону. Для фонів, також можна писати скрипти. Напишіть скрипт для зміни фону.



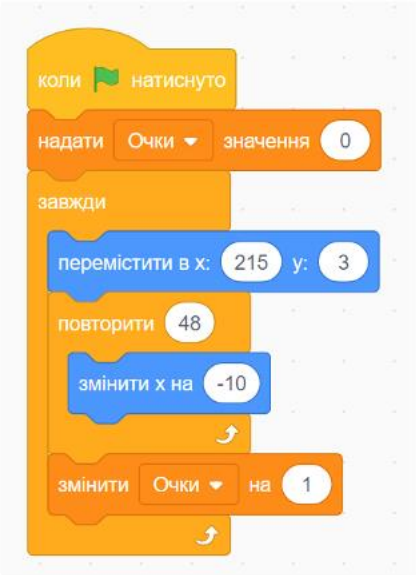


Підрахунок очків. Створіть змінну і назвіть її «очки».

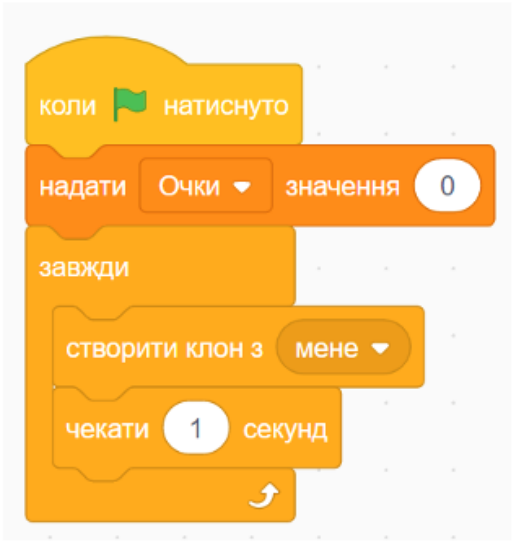


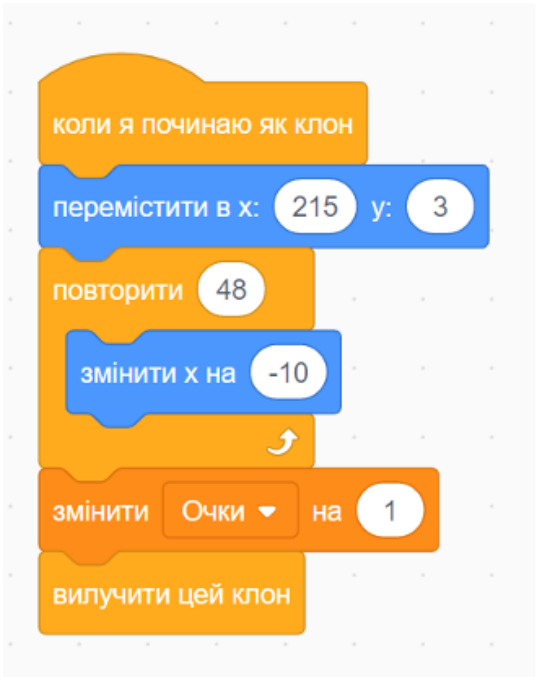


Підрахунок очків. Доповніть скрипт кактуса блоками з розділу «змінні».

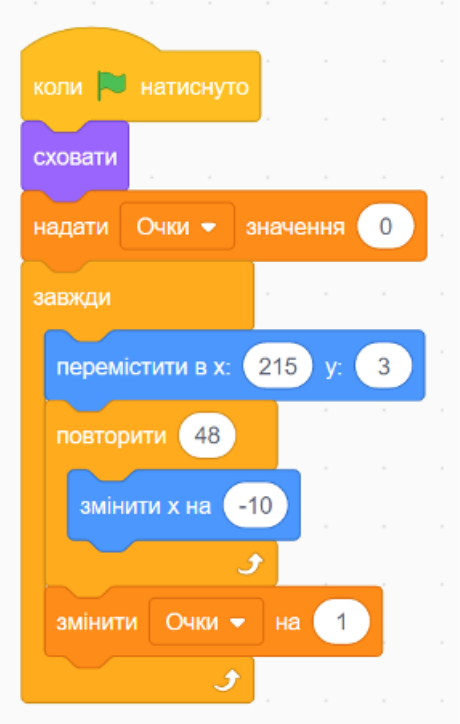


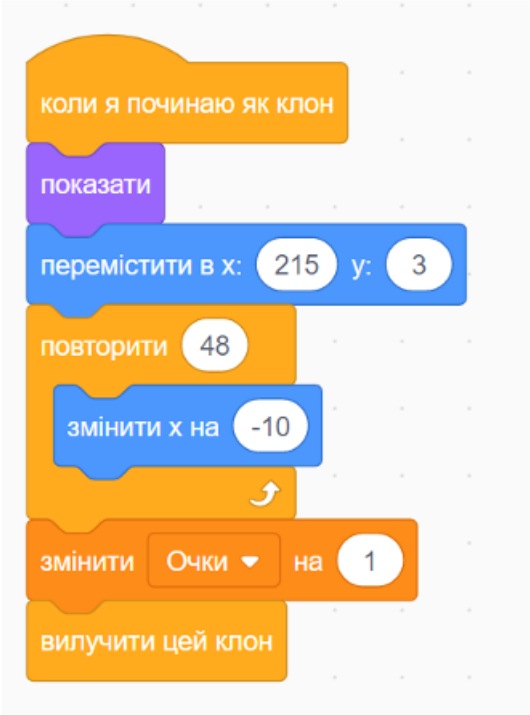
Кількість і інтервал перешкод. Для створення нових перешкод будемо використовувати клонування. Замініть скрипт перешкоди.



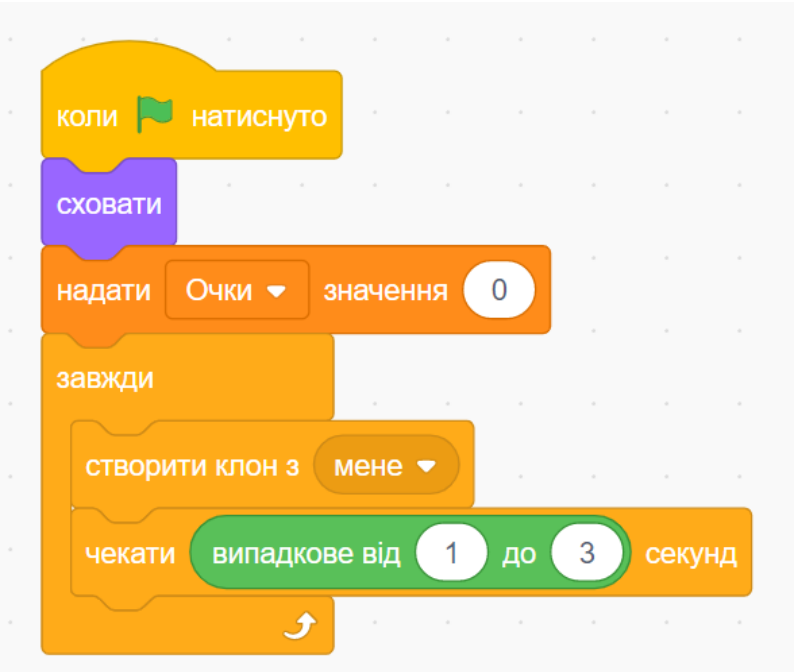


Сховайте оригінал перешкоди і показуйте тільки клони.

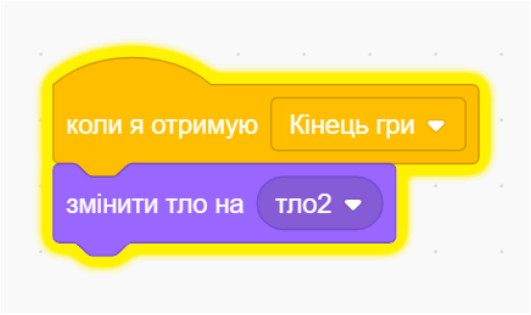




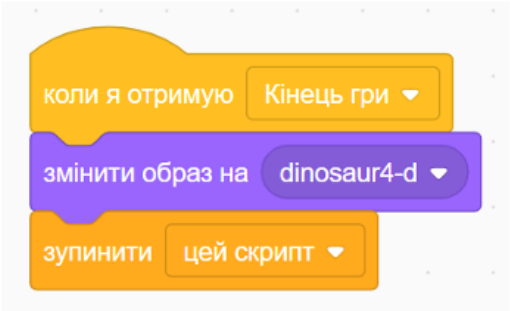
Для зміни інтервалу, потрібно замінити «чекати 1 секунду» на випадкове число.

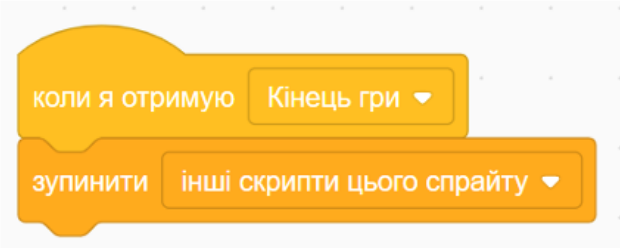


Коли гра зупиняється, видаляються всі клони. У скриптах фону видаліть «стоп все».

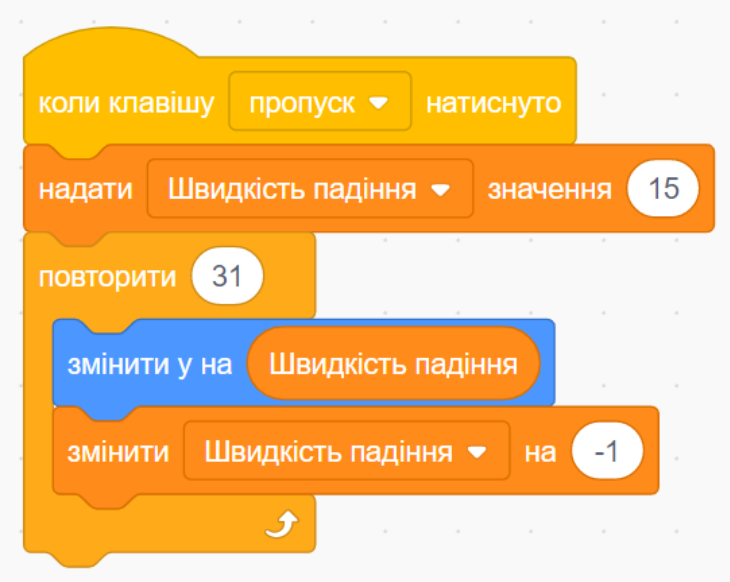


Додайте зупинку гри в скрипти динозавра і перешкоди.

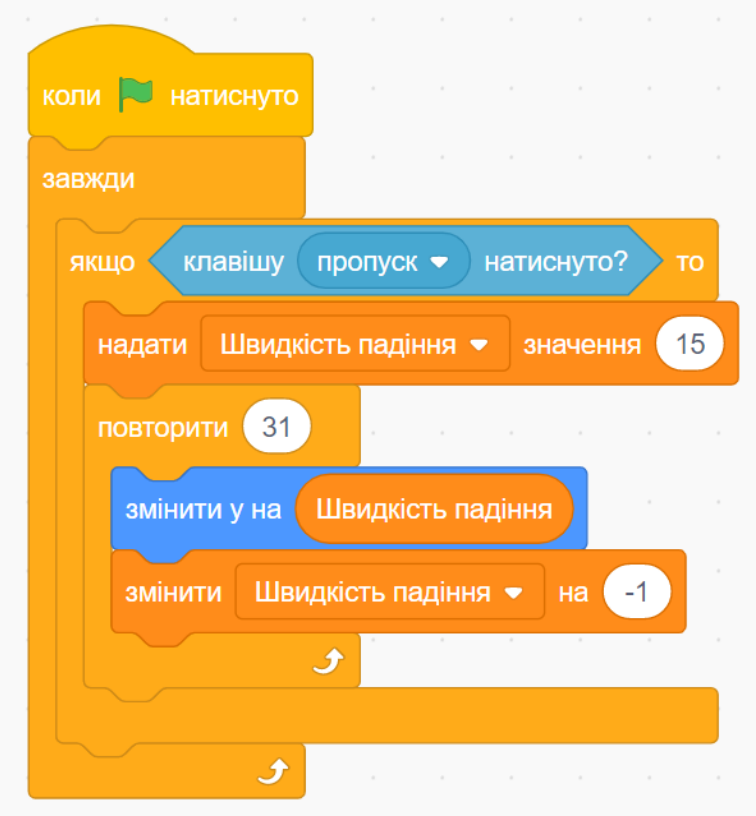




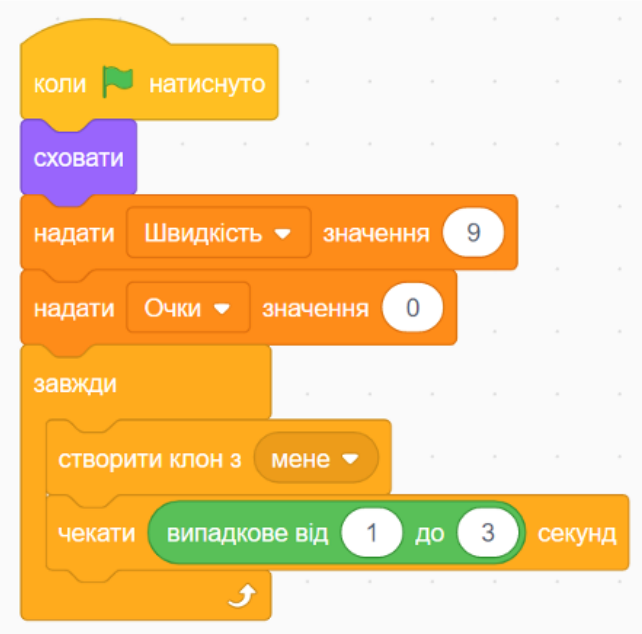
Смикання стрибку у динозавра. Введіть змінну «швидкість падіння». Створіть цикл від 15 до -15.

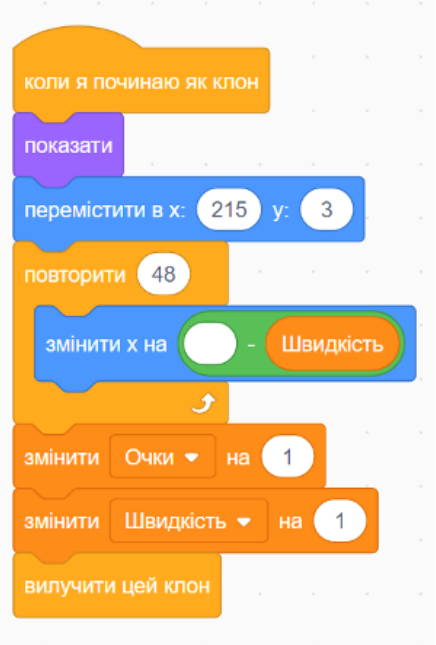


Динозавр стрибає після кінця гри. Змініть скрипт, щоб пробіл працював, тільки тоді, коли натиснуто прапорець.



Збільшення швидкості гри. Для прискорення гри створіть змінну «швидкість».





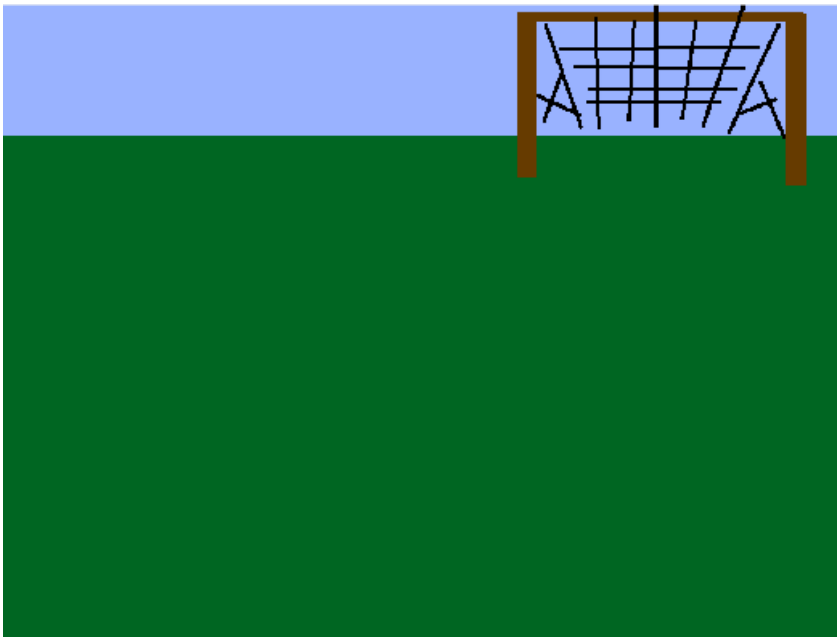
Завдання для самостійної реалізації:

1. Додайте «життя» головному персонажу.
2. Додайте ще одну перешкоду.

Завдання для самостійного виконання

1. Створіть гру «Пенальті».

За допомогою конструктора малювання намалюйте фон, футбольне поле, ворота. Приклад поля на рисунку.



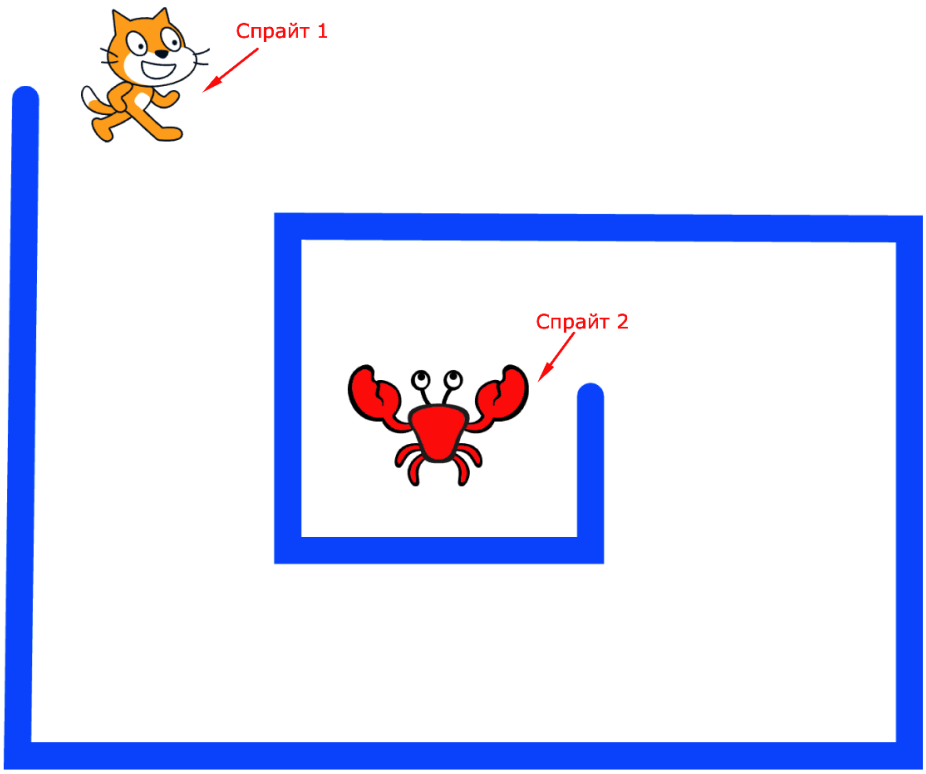
Додайте таких героїв-спрайтів: головний герой-спрайт, який буде виконувати пенальті, воротар-спрайт, який буде ловити м’яч, та персонаж-спрайт м’яч.

Створіть гру в якій головний герой буде бити пенальті у ворота суперника. Додайте героям різні костюми, щоб анімувати рух персонажів. Керування персонажами має здійснюватися різними обраними вами клавішами на клавіатурі, наприклад стрілками для головного персонажа та WSAD для воротаря.

Головний персонаж має спробувати забити м’яч у ворота суперника. Якщо в нього це виходить то +1 бал для головного героя, якщо ж м’яч піймав воротар то на головному екрані з’являється повідомлення «Game Over».

2. Створіть гру «Лабіринт».

За допомогою конструктора малювання намалюйте фон, лабіринт. Додайте 2 персонажі, які розмістіть так як на малюнку. Приклад на рисунку.



Додайте таких героїв-спрайтів: Спрайт 1 – персонаж, який буде визволяти персонажа, Спрайта 2, з лабіринта.

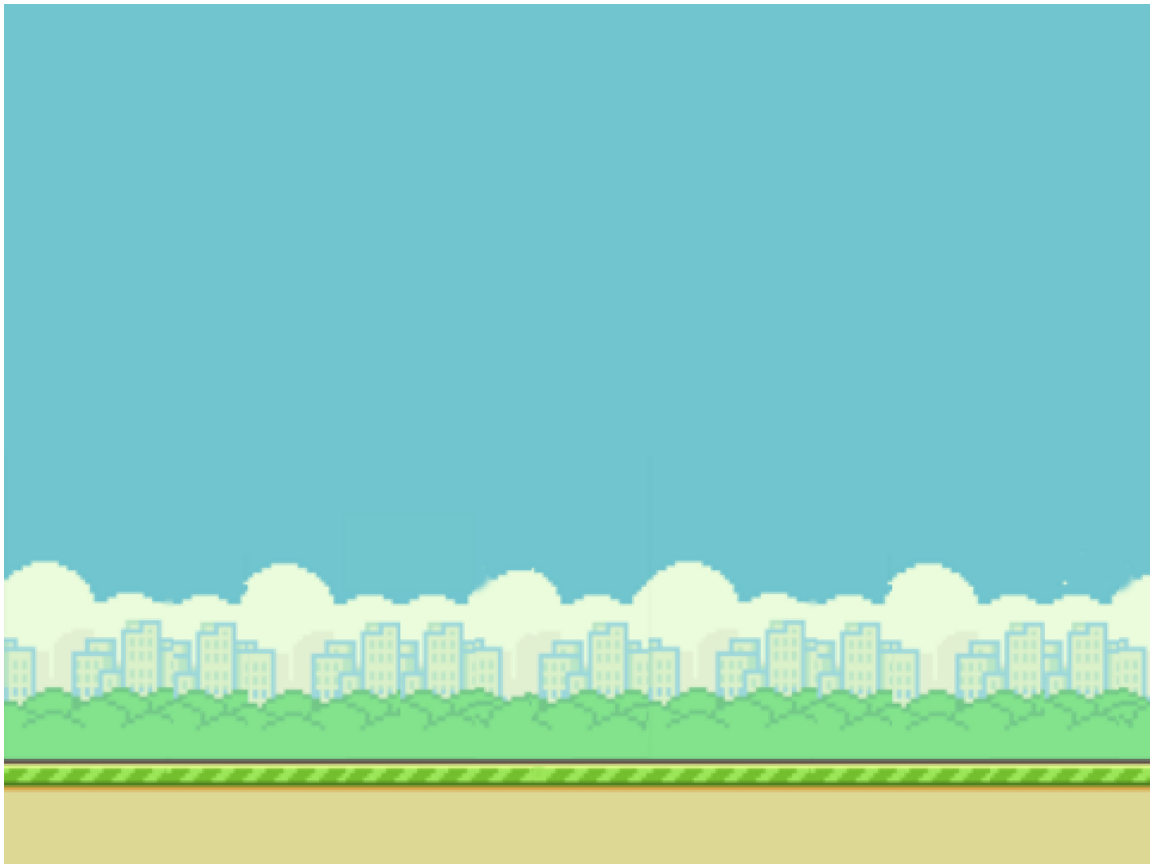
Створіть гру в якій головний герой, Спрайт 1, буде подорожувати лабіринтом, щоб знайти і визволити Спрайта 2.

Умови проходження гри: Спрайт 1 подорожуючи лабіринтом не має торкатись стінок лабіринту, якщо він торкається синьої лінії то гра зупиняється та персонаж вигукує «О ні, я програв!». Коли Спрайт 1 торкається Спрайта 2 то висвітлюється повідомлення про перемогу та Спрайт 2 починає підскакувати на місці та вигукувати «Мене врятували». Спрайт 1 має керуватись курсором мишки користувача.

3. Створіть гру «Flappy Bird».

Знайдіть в інтернеті фонове зображення знаменитої гри ***Flappy Bird***. Приклад на рисунку. Додайте спрайт-пташку, або ж будь-який інший спрайт,

який буде головним героєм нашої гри. Намалюйте або додайте предмети, труби, тобто перешкоди, через які буде проходити наш головний герой.



Створіть гру-прототип гри ***Flappy Bird.*** Головний герой постійно падає, щоб не дати йому впасти користувач клікає лівою кнопкою миші. Крім того головному герою потрібно оминути перешкоди, труби, які з'являтимуться на його шляху. З кожним проходженням таких перешкод головний герой отримує+1 бал. Якщо герой торкається труби на екрані з'являється повідомлення «Game Over». Та кнопка «Start new game». Швидкість руху труб довільна але із змогою її корегувати.

4. Створіть гру «Арканоїд»

Створіть проєкт гри, під час якої гравець контролює невелику платформу-дошку, яку можна пересувати горизонтально від однієї стінки до іншої, підставляючи її під м'яч, запобігаючи його падіння вниз. Удар м'яча по банану призводить до зникання банана та додавання балів. Виграє той, хто набере найбільшу кількість балів. Приклад на рисунку [3].

5. Створіть гру «Метелики»

Створіть проєкт гри, коли виконавець-сачок рухається за вказівником миші та ловить метелика за допомогою клавіші пропуск. Метелик з’являється у випадковому місці лугу і, коли його накриває сачок, він зменшується у розмірі та зникає. У цей час до змінної «Метелик» додається один бал [3].

6. *Створіть гру «Політ Баби-Яги»*

Створити проєкт, в якому «імітується» політ Баби Яги завдяки руху предметів, які знаходяться навколо, тобто об’єкти рухаються з одного краю до іншого, повторюючи дію за допомогою Завжди. Додайте до проєкту Зірки, та забезпечте і для них рух, враховуючи, що вони знаходяться на небі. Додайте до скрипту можливість керувати Бабою Ягою при натисканні клавіш «вгору-вниз» та додайте можливість «перестрибування» дерева [3].

7. Створіть гру «Вгадай число»

Комп’ютер випадковим чином породжує число у межах від 0 до 100, а гравець вгадує його. На пропозиції гравця комп’ютер повідомляє: «Мало», «Багато» чи «Ви вгадали!!!» залежно від взаємного розташування числа-здогадки і випадкового числа [3].

8. Створіть гру «Генератор кубиків»

Комп’ютер випадковим чином породжує число у межах від 0 до 100, а гравець вгадує його. На пропозиції гравця комп’ютер повідомляє: «Мало», «Багато» чи «Ви вгадали!!!» залежно від взаємного розташування числа-здогадки і випадкового числа. Спрайти та фон для реалізації гри намалюйте, знайдіть в мережі Інтернет чи оберіть з готового переліку [3].

9. Створіть гру «Хрестики-нулики»

В хрестики-нулики грали всі – від найменших до дорослих і навіть бабусі і дідусі. Цій грі вчили кожного з нас батьки ще в дитинстві. На сцені розміщені дев’ять спрайтів, які змінюють свої образи в залежності від того, який гравець робить хід. Спрайт має три образи – білий квадрат, хрестик, нулик. Почати гру потрібно, поставивши хрестик. Противник після цього поставить в довільній клітинці нулик. Ну і так далі. Переможцем стане той, чиї фігурки вишикуються в колону горизонтально або вертикально, а також по діагоналі. Якщо перемагають хрестики, то з’являється повідомлення «Хрестики виграли». Якщо перемогли нулики, то – «Нулики виграли». Інакше – нічия [3].

10. Створіть гру «Раллі»

Створіть проєкт під час, якого гравець рухає автомобілем клавішами управління курсором. Автомобіль має рухатись шляхом і не виїжджати на поле. Стрілка вгору та вниз регулює швидкість авто відповідно збільшує та зменшує. На табло виведені дані про швидкість авто, таймер руху та кількість кіл пройдених гравцем [3].

Увага! Спрайти до всіх завдань обирати з бібліотеки спрайтів або ж шукати у мережі Інтернет.

Розділ 3. PYTHON

3.1. Основи мови програмування Python

3.1.1. Вступ до мови програмування Python

Автором мови Python є нідерландський програміст Гвідо ван Россум. На той час він був співробітником голландського інституту CWI і працював над мовою ABC [en], яка мала стати максимально простою і зрозумілою декларативною мовою. Мова ABC була достатньо цікавою, але вона мала значний недолік: нею ніхто не користувався. Тому в грудні 1989 року Гвідо розпочав свій «хобі проєкт» Python, в основу якого лягли окремі елементи мови ABC (фактично, Python створювався як спроба виправити помилки, допущені при проєктуванні ABC). Датою появи мови Python вважається 20 лютого 1991 року.

До недавнього часу Гвідо ван Россум був не лише автором мови, а і «доброзичливим пожиттєвим диктатором» («Benevolent Dictator For Life») тобто мав право приймати остаточні рішення щодо мови. Проте 12 липня 2018 року він вирішив залишити цей «пост».

Як зазначає сам Гвідо ван Россум, мова названа на честь британського телешоу «Літаючий цирк Монти Пайтона» (англ. Monty Python’s Flying Circus), хоча інколи мову називають «Пітон».

З’явившись порівняно пізно, Python створювався під впливом багатьох мов програмування:

- ABC – на відміну від значної кількості мов програмування, в мові ABC відразу використовувалися відступи (поля) замість операторних дужок для групування операторів. Такий же принцип був обраний і для Python;
- Modula-3 – популярна в академічних кругах, але майже без практичного програмування мова була однією з перших, де були винятки (try … except), які і були взяті для Python;

- C, C++ – для Python були взяті деякі синтаксичні конструкції (як пише сам Гвідо ван Россум – він використовував найбільш несуперечливі конструкції з C, щоб не викликати неприязнь у C-програмістів до Python);
- Smalltalk – елементи об’єктно-орієнтованого програмування;
- Lisp – окремі риси функціонального програмування;
- Fortran – зрізи масивів, комплексна арифметика;
- Miranda – спискові вирази;
- Java – модулі logging, unittest, threading (частина можливостей оригінального модуля не реалізована), xml.sax стандартної бібліотеки, спільне використання finally та except при обробці винятків, використання @ для декораторів;
- Icon – генератори.

На даний момент існують дві актуальні версії мови: Python 2.x (остання версія на момент написання – 2.7.18) і Python 3.x (остання версія на момент написання – 3.12.0). Це пов’язане з тим, що за свій час «життя» мова Python 2.x акумулювала певну кількість недоліків, які не можна було виправити, не зламавши чийсь код. Основний недолік пов’язаний з рядками (спочатку рядки були байтовими, потім було додано підтримку Юнікоду (Unicode) з неявним приведенням).

Тому в більшості випадків Python 2.x використовується для забезпечення сумісності з раніше написаними програмами. Проте підтримка гілки 2.x буде завершена в січні 2020 року.

Python 3.0 обернено не сумісний з попередньою серією 2.x. Код Python 2.x швидше за все буде видавати помилки при виконанні в Python 3.x.

Динамічна типізація Python, разом з планами зміни декількох методів словників, робить механічне портування з Python 2.x в Python 3.0 дуже складним.

Тому нові проєкти варто створювати на Python 3.x, якщо немає явних причин використовувати Python 2.x [12].

Python – це високорівнева інтерпретована мова програмування, на відміну від C або C++, є прикладом компільованої мови програмування.

Назва Python стосується, як мови програмування, так і інтерпретатора, тобто комп'ютерної програми, яка зчитує початковий код (написаний на Python) та виконує інструкції (команди) [6].

Python – це багатоплатформова мова програмування. Це означає, що програми на Python можна запускати на різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, базами даних, вебсторінками, різними форматами даних, для створення графічного інтерфейсу (GUI) програм тощо. Програми, написані на мові програмування Python, зазвичай є невеликими скриптами, проте можуть бути складними системами.

Один з можливих недоліків Python – це швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній код (байт-код), а потім виконується інтерпретатором. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C або C++.

Утім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків (програм) швидкість розробки важливіша від швидкості виконання, а програми на Python зазвичай пишуться набагато швидше. Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор [6].

Синтаксис мови Python мінімалістичний і гнучкий. Цією мовою можна складати прості й ефективні програми. Стандартна бібліотека для цієї мови містить багато корисних функцій, що значно полегшує процес створення програмних кодів.

Мова Python підтримує декілька парадигм програмування, включаючи структурне, об’єктно-орієнтоване й функціональне програмування. І це далеко не весь список.

На сьогодні Python використовується при реалізації найрізноманітніших проєктів, серед яких:

- розробка сценаріїв для роботи з веб- та Інтернет-програмами;
- мережеве програмування;
- засоби підтримки технологій HTML і XML;
- програми для роботи з електронною поштою й підтримки Інтернет-протоколів;
- програми для обслуговування найрізноманітніших баз даних;
- програми для наукових розрахунків;
- програми з графічним інтерфейсом;
- створення ігор і комп’ютерної графіки та багато іншого [4].

Для написання комп’ютерних програм використовують текстові редактори або інтегровані середовища розробки (ICP), які включають в себе різні інструменти для роботи з кодом, такі як: інтерактивний інтерпретатор, відлагоджувач тощо.

Текстові редактори та ICP для Python:

1. IDLE – це стандартний редактор Python. Встановлюється разом з Python для користувачів Windows та окремим пакунком для користувачів Linux.

2. PyCharm – це інтегроване середовище розробки мови програмування Python. Підтримує веброзробку на Django.

3. Notepad++ – це безкоштовний текстовий редактор коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

4. Sublime Text 3 – це кросплатформовий текстовий редактор вихідних кодів програм та інтегроване середовище розробки. Підтримує плагіни, розроблені за допомогою мови програмування Python [6].

5. Replit – це інтегрований онлайн-компілятор з широким функціоналом, який працює з різними мовами програмування, включаючи Python, JavaScript, C++ та інші [9].

6. Stackless – патчі до CPython, який надає розширені можливості багатопотокового програмування і значно більшу глибину рекурсії;

7. PyPy – реалізація Python, написана на RPython. В PyPy вбудований трасуючий JIT-компілятор (Just-in-time compilation), який може перетворювати код Python в машинний код під час виконання програми [12].

8. PyScripter – некомерційне середовище розробки на мові Python [4]

3.1.2. Інсталяція Python

Щоб встановити Python в системі Microsoft Windows виконайте такі дії:

- перейдіть на офіційний сайт проєкту https://www.python.org/ (рис. 3.1.2.1.).

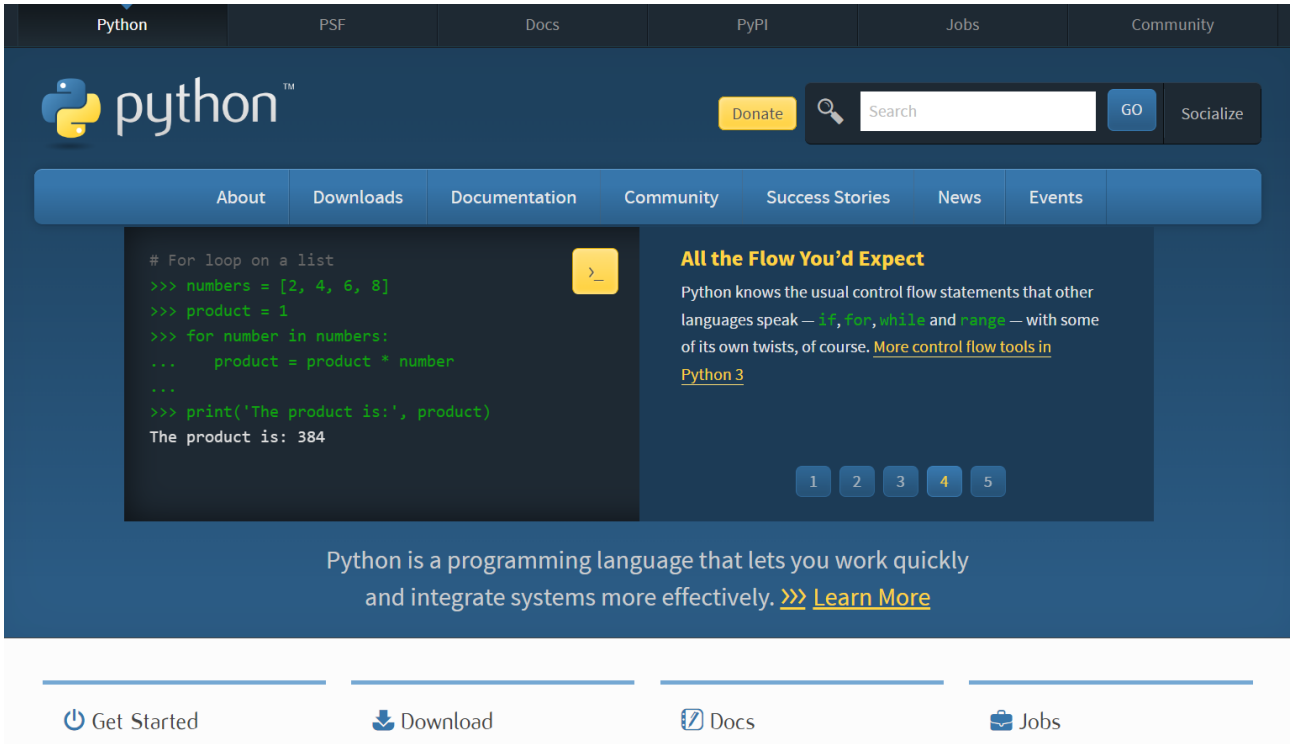


Рисунок 3.1.2.1. Головна сторінка сайту Python

- у вкладці Downloads натисніть Python 3.x.x для завантаження останньої версії або ж перейдіть до All releases та завантажте потрібну Вам версію;

- далі, у залежності від налаштувань вашого браузера, відбудеться завантаження інсталяційного пакету або вам перед початком скачування потрібно буде обрати папку для завантаження;
- після завершення завантаження відкриваємо папку та запускаємо файл інсталяції;
- ставимо прапорець біля пункту «Add Python to PATH» та обираємо «Install Now»;
- у вікні, що з'явиться натискаєте кнопку «Так», дозволяючи зробити зміни на комп'ютері;
- чекаєте доки завершиться процес інсталяції та натискаєте кнопку «Close».

Можете приступати до використання Python [24].

3.1.3. Інтегровані середовища розробки IDLE, PyCharm, Replit. Перша програма

Середовище IDLE

При інсталяції Python також встановлюється інтегроване середовище для розробки та навчання *IDLE* (Python's Integrated Development and Learning Environment). Після запуску програми Python відкриється вікно командної оболонки Python (рис. 3.1.3.1), яка входить до IDLE, а три знаки «більше» (>>>) називаються запрошенням. Після запрошення можна вводити різні команди.

У IDLE є два види вікон: вікно програми (рис. 3.1.3.2) та вікно консолі (рис. 3.1.3.1). У вікні програми можна писати, редагувати, зберігати та запускати програмний код, а результат виконання можна переглянути у вікні консолі. У вікні консолі також можна відразу виконувати команди Python.

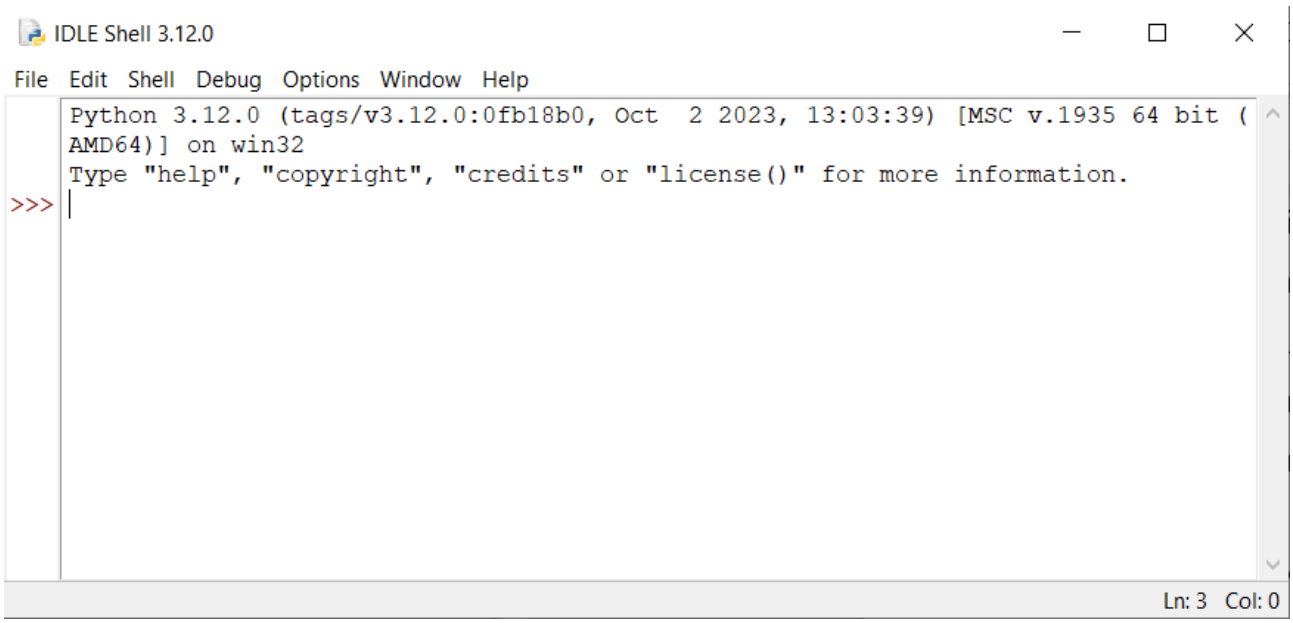


Рисунок 3.1.3.1. Вікно консолі IDLE Python

Вікно програми добре підходить для великих програм, які потрібно зберігати та редагувати. Це простіше, ніж друкувати команди кожного разу, коли вони знадобляться. Вікно консолі ідеально для експериментів – наприклад, якщо потрібно розібратися, що робить команда. Крім того, консоль можна використовувати як калькулятор. Однак, якщо якісь команди потрібно повторювати, краще використовувати вікно програми [24].

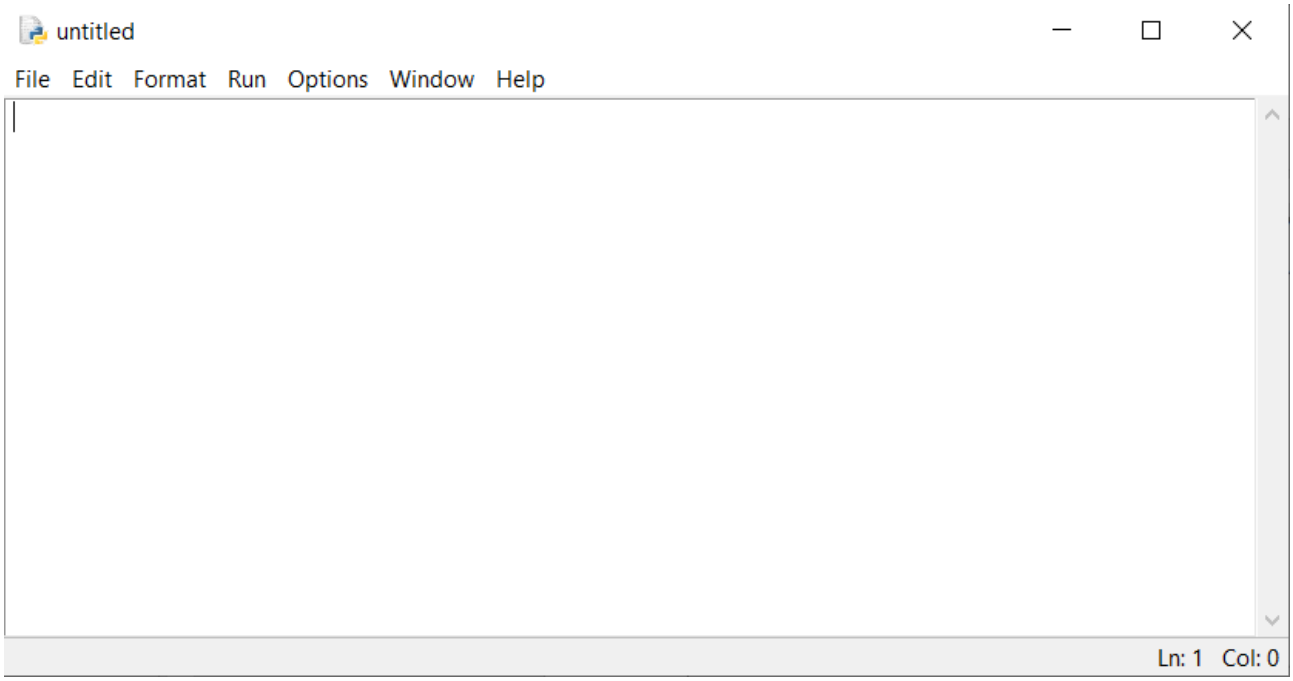


Рисунок 3.1.3.2. Вікно програми IDLE Python

PyCharm

PyCharm це кросплатформне, потужне інтегроване середовище розробки. PyCharm був розроблений компанією JetBrains – відомою міжнародною компанією, що спеціалізується на розробці інструментів для програмування різними мовами, як-от Java, Kotlin, C#, F#, C++, Ruby, Python, PHP, JavaScript і багатьох інших.

PyCharm надає розробникам безліч ключових функцій, які значно спрощують процес програмування. Ось кілька з них:

1. Налагодження коду. Потужний налагоджувач дає змогу знаходити та виправляти помилки, встановлюючи точки зупинки та аналізуючи значення змінних.

2. Рефакторинг. Інструменти рефакторингу допомагають змінювати структуру коду без втрати функціональності.

3. Підтримка систем контролю версій. Інтеграція з Git, Mercurial та іншими системами дає змогу зручно працювати з історією змін.

4. Автодоповнення коду. Інтелектуальне автодоповнення прискорює процес написання коду і знижує ймовірність помилок.

5. Інспектування коду. Статичний аналіз коду допомагає виявляти потенційні помилки та підтримувати високу якість коду.

6. Інтеграція з віртуальними оточеннями. PyCharm надає зручний інтерфейс для роботи з віртуальними оточеннями Python. Ви можете створювати та керувати різними оточеннями, що дозволяє ізолювати залежності та бібліотеки для кожного проєкту.

Плагіни для PyCharm – це додаткові розширення, які дають змогу додати нову функціональність у середовище розробки. Ось кілька прикладів популярних плагінів та їхній внесок у розширення функціональності:

Code Glance – додає міні-карту коду для зручної навігації по файлах.

Rainbow Brackets – розфарбовує дужки, роблячи код більш структурованим.

Django Support – забезпечує розширену підтримку для Django-проектів.

GitToolBox – покращує інтеграцію з системою контролю версій Git.

Markdown Support – надає підсвічування синтаксису та зручний перегляд Markdown-файлів.

Database Tools – забезпечує роботу з базами даних прямо із середовища розробки.

BashSupport – підтримка синтаксису й автодоповнення для Bash-скриптів.

PlantUML Integration – дає змогу малювати діаграми UML прямо в PyCharm [37].

Replit

Replit – це інтегрований онлайн-компілятор з широким функціоналом, який працює з різними мовами програмування, включаючи Python, JavaScript, C++ та інші. Він дає змогу користувачам писати код і створювати програми та вебсайти за допомогою браузера.

Першою важливою перевагою є те, що онлайн-IDE не потребує встановлення на персональний комп’ютер, для того, щоб почати працювати достатньо лише мати доступ до мережі Інтернет, крім того, це дає змогу почати процес кодингу незалежно від потужності апаратного забезпечення користувача та зменшити час витрачений на встановлення важких десктопних програмних засобів.

Щоб створити свою першу програму потрібно ввійти на ресурс через обліковий запис Google, Github чи Facebook, це надасть змогу зберігати всі свої програмні проєкти в одному акаунті, доступ до якого можливий як з персонального комп’ютера так і з будь-якого смартфона чи планшета, адже з сервісом можна працювати завантаживши однойменний додаток в Google Play чи Apple Store.

Ще однією перевагою є те, що онлайн-IDE підтримує більше 40 різних мов програмування, а синтаксис коду будь-якої мови виділяється різними кольорами, що, безперечно, сприяє кращому візуальному сприйнятті інформації.

Також у Replit доступна функція спільної роботи над кодом з чатом для обговорення.

Крім того програмувати в Replit дуже зручно, адже як тільки ви почнете писати певну функцію сервіс сам автоматично запропонує дописати один з її варіантів, більш того вікно програми та вікно консолі розміщені в одному робочому просторі, це дає змогу пришвидшити написання коду та оптимізувати свою роботу [9].

Перша програма

Традиційно при вивченні нової мови програмування першою програмою є програма-вітання «Hello, World!». Ось з неї і почнемо. Для цього:

- відкрийте IDLE;
- після >>> введіть print(‘ Привіт, світ! ‘);
- натисніть клавішу <Enter>;
- після виконання у новому рядку повинен з’явитися напис (рис. 3.1.3.3) [24].

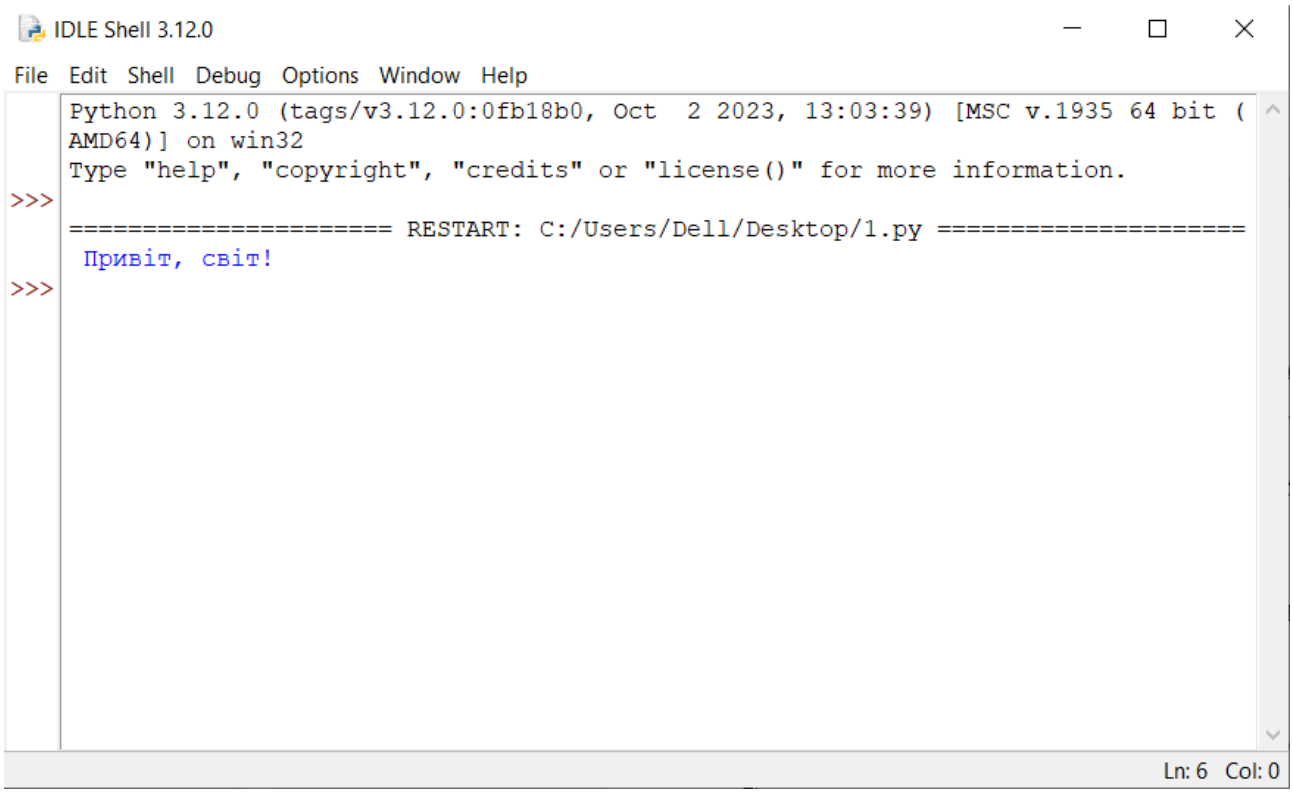


Рисунок 3.1.3.3. Програма «Привіт, світ!»

Давайте детальніше розглянемо, що робить Python, коли Ви запускаєте на виконання готову програму з розширенням *.ру. Виявляється, навіть для такої простої програми Python виконує досить серйозну роботу:

print (‘ Привіт, світ! ‘)

При виконанні цього коду виводиться наступний текст:

Привіт, світ!

Розширення *.ру в назві файлу вказує на те, що файл є програмою на Python. Редактор запускає файл в інтерпретаторі Python, який зчитує програму та визначає, що означає кожне слово в програмі. Наприклад, коли інтерпретатор виявляє слово print, він відображає текст, укладений у круглі дужки.

Коли ви пишете програму, редактор виділяє різні частини програми. Наприклад, він розуміє, що print – це ім’я функції, і друкує це слово одним кольором. З іншого боку, «Привіт, світ!» не є кодом Python, тому цей текст виділено іншим кольором. Цей механізм підсвічування кольорів синтаксису, дуже допоможе Вам, коли Ви почнете програмувати самостійно [34].

3.1.4. Початкові відомості про синтаксис мови Python

Синтаксис мови Python, як і сама мова, доволі простий.

- Кінець рядка є кінцем інструкції (крапка з комою не потрібна).
- Вкладені інструкції об’єднуються в блоки за величиною відступів.
- Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. І про читабельність коду не забувайте.
- Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробіли.
- Вкладені інструкції в Python записуються за одним і тим же шаблоном, а саме – основна інструкція завершується двокрапкою, слідом за якою розташовується вкладений блок коду, зазвичай з відступом під рядком основної інструкції.

- Іноді декілька інструкцій можна записувати в одному рядку, розділяючи їх крапкою з комою але краще цього не робити.
- Є можливим запис однієї інструкції у кількох рядках. Для цього її достатньо взяти у круглі дужки.

З іншими особливостями синтаксису мови Python будемо знайомитися пізніше під час її вивчення [24].

Коментарі

Коментарі призначені для вставки пояснень в текст програми. Інтерпретатор їх повністю ігнорує. Всередині коментаря може розташовуватися будь-який текст, включаючи інструкції, які виконувати не слід. Пам'ятайте, коментарі потрібні програмісту, а не інтерпретатору Python. Вставка коментарів до коду дозволить через деякий час швидко згадати призначення фрагмента коду.

У мові Python присутній тільки однорядковий коментар. Він починається з символу #. Однорядковий коментар може починатися не тільки з початку рядка, а й розташовуватися після інструкції. Якщо символ коментаря розмістити перед інструкцією, то вона також не буде виконана. Щоб продемонструвати це скористаємося файлом. Створимо його за допомогою інтегрованого середовища IDLE та збережемо під ім'ям Komentar, а потім додамо рядки коду, які наведено нижче:

```
print("Привіт, Світ!")
# Це звичайний коментар
```

або

```
print("Привіт, Світ!") # Це звичайний коментар
```

Якщо ми виконаємо програму, то побачимо, що додані коментарі на результат не вплинули [24].

Контрольні запитання

1. Що таке IDLE?
2. Які є два види вікон у IDLE? Розкажіть про їх призначення.

3. Як можна використовувати вікно Python Shell?

4. Яка повинна бути виставлена мова при використанні у середовищі Python Shell комбінацій клавіш?

5. Чи потрібна крапка з комою у мові Python в кінці інструкції?

6. Яким чином у мові програмування Python вкладені інструкції об'єднуються в блоки?

7. Чи є можливим у Python запис однієї інструкції у кількох рядках?

8. Яке призначення коментарів у мові Python?

9. Які правила додавання коментарів?

10. Які середовища програмування мовою Python Вам відомі?

Практичні завдання

За допомогою інтегрованого середовища IDLE обчисліть значення виразу

$$\frac{26^3}{4+3^4} - \frac{54-2^{-2}}{6\cdot\sqrt{4}}$$

1. В даному прикладі варто звернути увагу на те, як мовою програмування Python записуються математичні дії піднесення до степеня та знаходження квадратного кореня з числа. Кінцевий вигляд готового коду для обчислення виразу матиме вигляд [23]:

```
>>> (26**3)/(4+3**4)-((54-2**(-2))/(6*(4**(1/2))))
202.29730392156864
```

Завдання для самостійного виконання

1. За допомогою інтегрованого середовища IDLE обчисліть значення виразів:

a) $\frac{7+9-4}{5}$;

b) $7{,}77 \cdot 27{,}8 - 348 : 34 + 10$;

c) $(45 - 67) \cdot 34 + 100$;

d) $(7{,}2 + 2{,}4)^7 : 5{,}6$.

2. За допомогою інтегрованого середовища IDLE обчисліть значення виразів [24]:

a) $\frac{17-9\cdot4}{10}+9$;

b) $7{,}77\cdot(27{,}8+3{,}4):20-10$;

c) $9{,}8\cdot(6{,}3-3{,}5^4)$.

3. Напишіть декілька рядків коду, які відображають ваше ім’я та поштову адресу. Ваша програма не потребує ніякого введення від користувача, тільки вивід на екран і більше нічого [39].

3.2. Знайомство з модулями в Python

3.2.1. Модуль turtle (Черепашка)

Графічний модуль turtle (Черепашка) створений для тих хто починає вивчати програмування. Він надає можливості для малювання різних фігур. При роботі з графікою turtle можна написати інструкції для віртуальної Черепашки, що повідомлятимуть їй про те, як переміщатися по екрану. Черепашка має ручку (перо), а користувач може проінструктувати черепашку (скласти відповідну програму) – яким чином скористатися цією ручкою для малювання ліній під час переміщення по екрану.

Використання графіки turtle дає можливість не тільки створювати малюнки та орнаменти за допомогою всього лише кількох рядків коду, але також дозволяє відстежувати рух черепашки, щоб зрозуміти, яким чином кожний рядок коду впливає на траєкторію переміщення черепашки. Це допомагає зрозуміти логіку програмного коду.

Виконавець Черепашка керується командами відносних («вперед-назад» і «вправо-вліво») і абсолютних («перейти у точку з координатами ...») переміщень. Він має «перо», що залишає слід на площині малювання (полотні). Перо можна підняти, тоді при переміщенні слід залишатися не буде. Крім того,

для пера можна встановити товщину і колір. Всі ці функції виконавця забезпечуються модулем turtle (документація з модуля turtle (Черепашка).

Наведений нижче код створює графічне вікно і поміщає перо («Черепашку») у початкове положення.

```
from turtle import*
reset()
exitonclick()
```

Команда *from turtle import** використовується для підключення модуля Черепашки. Команда *reset()* повертає Черепашку (стрілочка у центрі) до початкового стану: очищається полотно, скидаються всі параметри, а Черепашка встановлюється в початок координат, дивлячись вправо. Виконання інструкції *exitonclick()* призводить до закриття графічного вікна при натисканні лівої кнопки мишки.

Ідея малювання полягає в переміщенні пера (Черепашки) у точки полотна з вказаними координатами або у зазначених напрямках на задані відстані, а також у проведенні відрізків прямих, дуг та кіл.

Поточний напрямок переміщення пера (що відповідає напрямку «вперед») вказується вістрям стрілки зображення Черепашки.

Список команд управління Черепашкою (і, відповідно, малювання), а також функцій, які забезпечуються модулем, можна отримати, набравши команду *help('turtle')*. Список цей досить довгий, а серед функцій є також математичні, оскільки вони можуть бути затребувані при обчисленні параметрів відрізків, дуг та кіл. Наведемо, команди Черепашки, які ми будемо використовувати у посібнику (табл. 3.2.1).

Таблиця 3.2.1

Синтаксис	***Призначення***	***Приклади використання***
Команди переміщення		
forward(n) **fd(n)** **(forward – вперед)**	Переміщає Черепашку вперед на n кроків (пікселів).	`forward(10)` `fd(50)`
back(n) **bk(n)** **(back – назад)**	Переміщає Черепашку назад на n кроків (пікселів).	`back(60)` `bk(70)`

right(k)
rt(k)
	Поворот направо (за годинниковою стрілкою) на k одиниць.	`right(90)`
		`fd(50)`
		`rt(90)`

left(k)
lt(k)
	Поворот наліво (проти годинникової стрілки) на k одиниць.	`fd(50)`
		`left(45)`
		`forward(100)`
		`lt(45)`

circle(r)	Малювання кола радіусом |r| точок з поточної позиції пера. Якщо r додатне, коло малюється проти годинникової стрілки, якщо від'ємне – за годинниковою стрілкою.	`circle(20)` `circle(-40)`
circle(r,k)	Малювання дуги радіусом |r| точок і кутом k одиниць. Варіант команди circle().	`circle(45,60)` `circle(-70,180)`
goto(x,y) **setpos(x,y)**	Переміщення пера (Черепашки) у точку з координатами (x, y) в системі координат полотна.	`goto(5,-45)` `setpos(-100,100)`
setx(x)	Встановити x-координату Черепашки.	`setx(50)`
sety(y)	Встановити y-координату Черепашки.	`sety(100)`
home()	Повернути черепашку додому – у точку, з координатами (0,0).	`home()`
setheading(angle)	Повернути Черепашку під кутом angle (90 – вгору, 0 – вправо).	`setheading(30)`
speed(v) **(speed – швидкість)**	Встановити швидкість черепашки. v має бути від 1 (повільно) до 10 (швидко), або 0 (миттєво).	`speed(1)`
delay(z)	Встановлює затримку малювання в мілісекундах. Чим більше z, тим повільніше рухається Черепашка. (z≥0)	`delay(25)`
Команди малювання		
down() **(down – вниз)**	Опускає перо (по замовчанню – перо опущене). Після цієї команди Черепашка почне залишати слід при будь-якому своєму пересуванні.	`home()` `down()`
up() **(up – вгору)**	Піднімає перо.	`up()`
dot(d[, 'color'])	Малює точку діаметра d кольору color. Параметр color необов'язковий.	`dot(30, 'red')`

width(n)
(width – ширина)
	Встановлює ширину пера Черепашки у n пікселів.	`fd(50)`
		`width(10)`
		`fd(50)`

color('color') **(color – колір)**	Встановлює колір Черепашки. Колір вказується у дужках англійською мовою у верхніх лапках – 'red'. (Див. Додаток 5. Перелік кольорів.)	`color('green')` `fd(50)`
color('color', **'fillstring')**	Дозволяє змінити кольори ліній та заливки.	

fillcolor('color')	Встановлює колір зафарбовуваної області.	`fillcolor('red')`

begin_fill() та end_fill()	Використовуються для малювання зафарбованих областей. Перед початком малювання вказується команда begin_fill(), а у кінці – end_fill().	`begin_fill()`
		`fd(100); lt(90)`
		`fd(100); lt(90)`
		`fd(100); lt(90)`
		`fd(100); lt(90)`
		`end_fill()`

hideturtle()	Сховати Черепашку.	`hideturtle()`
showturtle()	Показати Черепашку.	`showturtle()`
Інформація про Черепашку		
position() **pos()**	Отримати поточні координати Черепашки.	`x,y=pos()` `print(x, y)`
xcor()	Отримати x координату Черепашки.	`b=xcor()`
ycor()	Отримати y координату Черепашки.	`y=ycor()` `print(y)`
Інші команди		
radians()	Встановлює одиниці вимірювання кутів у радіанах.	`radians()` `lt(1.7)`
degrees()	Встановлює одиниці вимірювання кутів у градусах (увімкнений за замовчуванням).	`degrees()` `lt(90)`
undo()	Відміняє попередню дію Черепашки.	`undo()`
reset() **(reset – скинути)**	Повертає Черепашку (стрілочка по центру) до початкового стану: очищається полотно, скидаються всі параметри, Черепашка встановлюється у початок координат, дивлячись вправо.	`reset()`
clear()	Очищає полотно. Черепашка залишається на попередньому місці.	`fd(50)` `clear()`

write(s, move, align, font)	Виводить текстовий рядок s у точці знаходження Черепашки. Параметр move може приймати значення True або False (по замовченню move=False). move=False – Черепашка залишається на місці. move= True –Черепашці переміщається у кінець тексту. Параметр align вирівнює текст відносно початкового положення Черепашки (left, center або right). Font – визначає характеристики шрифту (Назва, Розмір, Тип).	`write('Всім привіт!!!',`
		`move=True,`
		`align="center",`
		`font=("Arial",`
		`45,`
		`"normal"))`

exitonclick()	Призводить до закриття графічного вікна при натисканні лівої кнопки мишки.	`from turtle import*`
		`reset()`
		`shape("triangle")`
		`exitonclick()`

done() **(done – зроблено)**	Остання команда у програмі для Черепашки. Не слід використовувати, якщо сценарій запускається з IDLE.	done()

shape()
(shape – форма)
	Дозволяє змінити форму Черепашки. Існують такі варіанти: “arrow” (стрілка), “turtle” (черепашка), “circle” (коло), “square” (квадрат), “triangle” (трикутник), “classic” (класичний).	`from turtle import*`
		`reset()`
		`shape("triangle")`
		`exitonclick()`

x,y = pos()	Поточні координати присвоюються змінним x та y.	`setpos(-100,100)`
		`x,y = pos()`
		`goto(x+30, y-70)`

setup(n,m) **(setup – установка, налагодження)**	Встановлює розміри екрану довжиною n пікселів та шириною m.	setup(1000,500)

3.2.2. Змінні. Типи даних у мові Python

Ідентифікатор Python – це ім’я, яке використовується для ідентифікації змінної, функції, класу, модуля або будь-якого об’єкта.

Ідентифікатор може містити тільки такі символи:

1. Літери в нижньому регістрі (від «a» до «z»).

2. Літери у верхньому регістрі (від «A» до «Z») (Python є регістрочутливою мовою програмування).

3. Цифри (від 0 до 9).

4. Нижнє підкреслення (_).

5. Не можуть збігатися з зарезервованими словами (див. табл. 3.2.2).

Таблиця 3.2.2

№	Назва	№	Назва	№	Назва	№	Назва	№	Назва
1	and	8	del	15	from	22	nonlocal	29	return
2	as	9	elif	16	global	23	not	30	True
3	assert	10	else	17	if	24	None	31	try
4	break	11	except	18	import	25	yield	32	while
5	class	12	False	19	in	26	or	33	with
6	continue	13	finally	20	is	27	pass	34	yield
7	def	14	for	21	lambda	28	raise		

Ідентифікатор не може починатися з цифри. Таким чином, A1 та a1 – два різних ідентифікатори в Python. Коректними є такі імена: var_a; var_a1. Зазвичай великими літерами в Python позначаються константи.

Код написаний мовою Python не потребує фігурних дужок для позначення блоків коду для визначення класів, функцій чи регулювання потоку. Блоки коду відокремлюються за допомогою, так званого рядкового відступу, якого потрібно жорстко дотримуватися (4 пробіли).

Наприклад:

```
if True:
 print("Істинна")
else:
 print("Хибне")
```

Однак, наступний блок коду згенерує помилку:

```
if True:
 print("Відповідь")
 print("Істинна")
else:
 print("Відповідь")
 print("Хибне")
```

Python приймає одиничні (‘), подвійні (‘‘) і потрійні (‘‘‘) лапки, щоб позначити рядкові літерали. Необхідною умовою є лише те, що відповідний рядок має починатися і закінчуватися одним типом лапок. Однак всередині рядка можна використовувати інші види лапок.

Потрійні лапки використовують для розбиття рядка на частини.

Наприклад:

```
var_word = ‘Слово’
var_sentence = ‘‘Речення’’
var_paragraph = ‘‘‘Параграф..
      текст параграфа...’’’
```

У міру збільшення обсягу програми рано чи пізно код стане складніше читати. Для підвищення зрозумілості коду його доповнюють коментарями на звичайній мові.

Уведення даних з клавіатури (починаючи з версії Python 3.x) здійснюється за допомогою функції input(). Якщо ця функція виконується, то потік виконання програми одразу ж зупиняється в очікуванні даних, які користувач повине ввести за допомогою клавіатури. Після уведення даних і натискання клавіші Enter функція input() завершує своє виконання та повертає результат, який є рядком символів, введених з клавіатури.

```
>>> input()
1234
'1234'
>>> input()
Hello World!
'Hello World!'
```

Дані повертаються у вигляді рядка, навіть при введені числа. Якщо потрібно отримати число, то результат виконання функції input() змінюють за допомогою функцій перетворення типів: int() або float().

```
>>> int(input('Введіть число: '))
Введіть число: 10
10
>>> float(input('Введіть число: '))
Введіть число: 10
10.0
```

Результат, що повертається функцією input(), зазвичай присвоюють певній змінній для подальшого використання. Функція print() має кілька аргументів (прихованих), які задаються за замовчуванням в момент виклику:

```
>>> help(print)
Help on built-in function print in module builtins:
```

```
print(...)
  print(value, ..., sep=‘ ‘, end=‘\n’, file=sys.stdout, flush=False)
...
```

де value – перераховуються об’єкти через кому, які потрібно вивести на екран, sep – рядок-розділювач між рядками (за замовчуванням стоїть пробіл), end – рядок, розміщений після останнього об’єкта (за замовчуванням – це перехід на новий рядок).

Отже, функція print() додає пробіл між кожним виведеним значенням або об’єктом, а також символ нового рядка в кінці:

```
>>> print(1,2,3,4,5)
1 2 3 4 5
>>> print(1,2,3,4,5, sep=‘:’)
1:2:3:4:5
```

Мови програмування також дозволяють визначати змінні.

Змінна – це не що інше, як зарезервоване місце в пам’яті для зберігання значень. Це означає, що при створенні змінної виділяється деякий простір пам’яті на ПК. Виходячи з типу даних змінної, інтерпретатор виділяє пам’ять і вирішує, що можна зберегти у виділеній пам’яті. Тому, призначаючи різні типи даних змінам, можна зберігати цілі числа, десяткові значення, логічні змінні або символи.

Змінні в Python – це просто імена, які є посиланням на значення в пам’яті комп’ютера. В Python символ «=» застосовується для присвоювання значення змінної. Операнд ліворуч від оператора присвоєння (=) – ім’я змінної, а операнд справа від нього – це значення, що зберігається в змінній.

Наприклад:

```
Variable_1 = 11
Variable_2 = 111.0
Variable_3 = "Привіт!"
print(Variable_1)
print(Variable_2)
```

```
print(Variable_3)
```

Тут 11, 111.0 та "Привіт!" – присвоєні значення для назв змінних Variable_1, Variable_2 та Variable_3, відповідно. Результатом будуть такі значення:

```
11
111.0
Привіт!
```

Присвоєння не копіює значення, воно прикріплює ім’я конкретного об’єкта, який містить безпосередньо дані [6].

В статичних мовах програмування вказується тип кожної змінної, який визначає, скільки місця змінна буде займати в пам’яті і що з нею можна виконати. Комп’ютер використовує цю інформацію, щоби скомпілювати програму у низькорівневу мову.

Оголошення типів змінних допомагає комп’ютеру знаходити помилки і працювати швидше, але це вимагає попереднього набору коду. Велика частина мов, таких як C, C++ та Java, вимагають оголошення типів для змінних.

Тип даних – це множина значень та множина операцій на ними. Тобто тип даних визначає можливі значення, їх зміст, операції над ними та способи їх зберігання.

Типізація – це операція призначення конкретного типу інформаційним сутностям. Для різних мов програмування існують різні види типізації: статична/динамічна та сильна/слабка (табл. 3.2.3).

Таблиця 3.2.3

Типізація	Статична	Динамічна
Сильна	C#, Java	Python, Ruby
Слабка	C	JavaScript, PHP

При *статичній типізації* тип даних визначається під час компіляції. Змінні, які не можуть змінити свого типу, є статичними. Наприклад, ціле число є цілим числом, раз і назавжди.

При динамічній типізації тип змінної визначається під час присвоєння їй значення. Якщо написати x = 5, то динамічна мова визначить, що 5 є цілим числом, тому змінна x має тип integer. Такі мови дозволяють досягти того ж результату, використовуючи при цьому меншу кількість рядків коду.

Проте, динамічні мови зазвичай повільніші, ніж статичні, але їх швидкість підвищується, оскільки інтерпретатори стають дедалі більш оптимізованими.

Тривалий час динамічні мови програмування застосовувалися виключно для написання коротких програм, які використовувалися для того, щоби підготувати дані для обробки більш довгими програмами, написаними вже на статичних мовах.

Сильна типізація означає, що виконання операцій при несумісності типів недопустне.

Натомість *слабка типізація* допускає виконання операцій над сутностями з різними типами, в результаті чого часто одержується непередбачуваний результат.

Відносний лаконізм Python дозволяє написати програму, яка буде набагато коротшою від свого аналога, написаного статичною мовлю.

В Python все (рядки, булеві значення, цілі числа, числа з рухомою комою, функції та різні інші структури даних) реалізовано як об’єкти. Це дозволяє Python бути стабільним, у порівнянні з іншими мовами.

Python є динамічною мовою із сильною типізацією, тобто тип об’єкта не зміниться, навіть якщо існує можливість змінити його значення.

Python включає в себе такі типи даних (рис. 3.2.2.1):

1. Текстовий: рядки (str).

2. Числовий: цілі (int), з рухомою комою (float), комплексі (complex).

3. Тип послідовностей: список (list), кортеж (taple), діапазон (range).

4. Тип картографування (mapping): словник (dict).

5. Тип наборів: набір (set), заморожений набір (frozenset).

6. Логічний тип: bool.

7. Бінарні типи: байти (bytes), байт-масив (bytearray), перегляд пам’яті (memoryview).

8. Тип даних None [6].

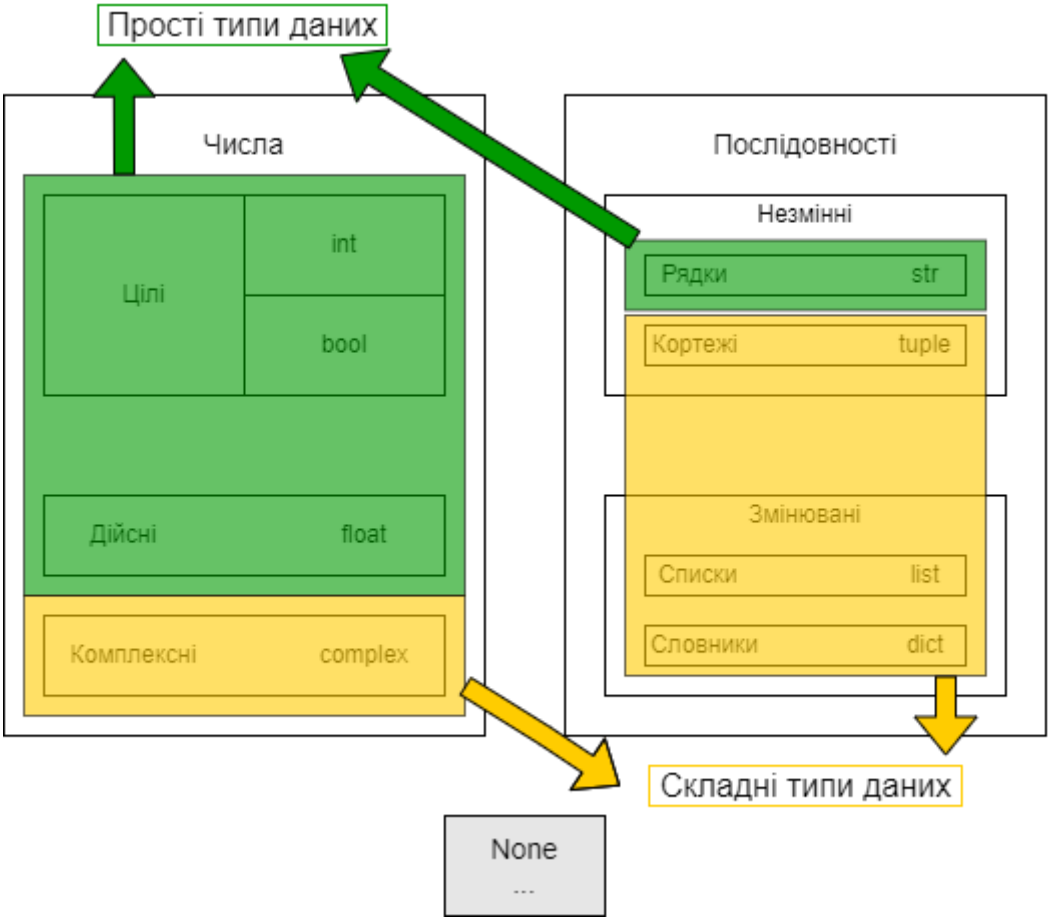


Рисунок 3.2.2.1. Типи даних мови Python

3.2.3. Оператори Python

Оператор в програмуванні – це спеціальний знак, який повідомляє транслятору про те, що потрібно виконати операцію з деякими операндами, також в книгах зустрічається термін інструкція програми.

Літерал (англ. literal – константа) – постійне значення певного типу даних, записане у вихідному коді комп’ютерної програми. У наступному прикладі, 44 і dog – літерали, а num і animal – змінні.

```
>>> num = 44
>>> animal = "dog"
```

Оператори використовуються для виконання операцій зі змінними та значеннями. Python ділить оператори на такі групи:

1. Арифметичні оператори.
2. Оператори присвоєння.
3. Оператори порівняння.
4. Логічні оператори.
5. Оператори ідентичності.
6. Оператори членства.
7. Побітові оператори.

Арифметичні оператори використовуються з числовими значеннями для виконання загальних математичних операцій.

Таблиця. 3.2.4

№	Оператор	Операція
1	+	сума
2	–	різниця
3	*	добуток
4	/	ділення
5	%	ділення по модулю
6	//	цілочислове ділення
7	**	піднесення до степеня

Оператори додавання, віднімання, множення і ділення не повинні викликати будь-яких складнощів.

При використанні декількох операторів рекомендується групувати вираження за допомогою дужок – операції всередині дужок виконуються першими.

Оператори присвоєння використовуються для присвоєння значень змінним. Всі вони, за винятком простого присвоювання, «=», є скороченими формами від довших виразів. Для ясності в таблиці представлені їх еквіваленти.

Таблиця. 3.2.5

№	Оператор	Еквівалент
1	=	x = y
2	+=	x += (x + y)
3	–=	x –= (x – y)
4	*=	x *= (x * y)
5	/=	x /= (x / y)
6	%=	x %= (x % y)
7	//=	x //= (x // y)
8	**=	x **= (x ** y)

У вище наведеній таблиці вище змінній з ім’ям x присвоюється значення, яке міститься в змінній з ім’ям y, і, таким чином, в змінній x буде зберігатися нове значення.

Оператори порівняння використовуються для порівняння двох значень.

Таблиця. 3.2.6

№	Оператор	Зміст
1	==	рівність
2	!=	нерівність
3	>	більше
4	<	менше
5	>=	більше або рівне
6	<=	менше або рівне
7	in	перевірка на входження до послідовності.
8	is	перевіряє, чи посилаються дві змінні на один і той же об’єкт.

Оператор подвійної рівності «==» здійснює порівняння двох операндів і повертає True (істина), якщо їх значення рівні, в іншому випадку повертає False (хиба). При цьому, якщо операндами є числові значення, і вони однакові, то вони рівні, а якщо символи, то порівнюються їх ASCII-коди.

Логічні оператори використовуються для об'єднання умовних операторів.

Таблиця. 3.2.7

№	Оператор	Операція
1	and	логічне «І»
2	or	логічне «АБО»
3	not	логічне «НЕ»

Логічні оператори працюють з операндами, які мають значення логічного (булева) типу, тобто True або False, або зі значеннями, які перетворюються в True або False.

Оператор логічне «І», (and), оцінює два операнда і повертає значення True, тільки якщо обидва операнди самі мають значення True, в іншому випадку повертає значення False. На відміну від оператора and, яким необхідно, щоб обидва операнда мали значення True, оператор логічне «АБО», (or), оцінює два операнда і повертає True, якщо хоча б один з них сам повертає значення True. В іншому випадку оператор or поверне значення False. Оператор «логічне НЕ», (not), унарний і використовується з одним операндом. Він повертає протилежне значення від того, яке мав операнд.

Оператори ідентичності (is та is not) використовуються для порівняння об'єктів, не коли вони рівні, а якщо це насправді один і той же об'єкт, з однаковим розташуванням в пам'яті та типом).

Оператори членства (in та not in) використовуються для перевірки наявності послідовності в об'єкті.

У мові Python можна працювати з окремими частинами байта, використовуючи так звані побітові оператори.

Таблиця. 3.2.8

№	Оператор	Назва	Приклад
1	|	«І»	1010 | 0101 = 1111
2	&	«АБО»	1010 & 1100 = 1000
3	~	«НЕ»	1010 ~ 0011 = 0100
4	^	Виключне «АБО»	1010 ^ 0100 = 1110
5	>>	Зсув вліво	0010 << 2 = 1000
6	<<	Зсув вправо	1000 >> 2 = 0010

Дуже важливо розрізняти типи даних, особливо при присвоєнні змінним значень, використовуючи користувацьке введення, оскільки за замовчуванням в ньому зберігається рядковий тип даних.

Рядкові величини не можуть бути використані для арифметичних виразів, і спроба скласти два рядкових значення просто об'єднує ці рядки, а не використовує операції над числами, наприклад: '4' + '7' = '47'.

Контрольні запитання

1. Для чого призначений модуль turtle (Черепашка)?
2. Яка інструкція використовується для підключення модуля Черепашки?
3. Яким чином можна отримати весь список команд управління «Черепашкою»?
4. Які команди з переміщення Черепашки вам відомі?
5. Перерахуйте відомі вам команди Черепашки для малювання?
6. З чого може складатися ім'я змінної у мові Python?
7. Що означає динамічна типізація?
8. Яким чином у Python здійснюється позиційне присвоювання?
9. Чи важливо враховувати регістр в іменах змінних у мові програмування Python?
10. Яке призначення операції присвоювання?
11. Які переваги вбудованих у мову Python типів?

12. Перерахуйте відомі вам типи даних, що можуть мати об’єкти у мові програмування Python 3?

13. Які числові типи підтримує Python 3?

14. В мові Python якого типу буде результат операції у якій беруть участь ціле і дійсне числа?

15. Як на мові Python записується математичний оператор «Ціла частина від ділення»?

16. Який оператор на мові Python записується наступним чином: a % b?

17. Як на мові Python записується математичний оператор «Піднесення до степені»?

18. Перерахуйте відомі вам оператори присвоювання у мові програмування Python? Як вони позначаються та виконуються?

19. Який пріоритет математичних операторів у Python? За допомогою чого його можна змінити?

Практичні завдання

Завдання №1

Створіть програму після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано прямокутник, розміри якого оберіть довільні (наприклад 250 і 150).

Технологія виконання

1. Запустіть IDLE.

2. Введіть текст програми.

```
from turtle import*
reset ()
fd (150)
rt (90)
fd (80)
rt (90)
fd (150)
rt (90)
fd (80)
```

```
rt (90)
exitonclick ()
```

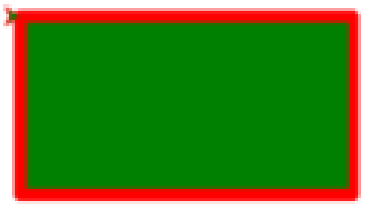
3. Запустіть програму на виконання. Збережіть програму під назвою Test.

4. У вікні Python Turtle Graphics перегляньте результати виконання.

5. За потреба відкоригуйте код програми.

Завдання №2

Створіть та збережіть на диску програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано прямокутник згідно зразка на рисунку, розміри оберіть довільні.



Технологія виконання

1. Запустіть IDLE. Для спрощення виконання відкрийте файл, що є результатом виконання завдання №1, та збережіть його під ім’ям Завдання №2.

2. Для збільшення товщини лінії та зміни кольору лінії потрібно після команди reset() додати команди width(7) та color(‘red’) відповідно. Щоб здійснити заливку прямокутника зеленим кольором після команди color(‘red’) необхідно вставити інструкцію fillcolor(‘green’) та begin_fill(), а перед командою exitonclick() інчтрукцію end_fill().

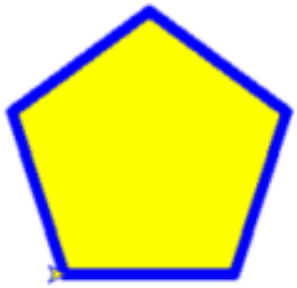
3. Збережіть внесені до файлу програми зміни.

4. Запустіть програму на виконання.

5. Перегляньте результати виконання та при потребі зробіть зміни у програмному коді.

Завдання №3

Створіть та збережіть на диску програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) п’ятикутник згідно зразка, наведеного на рисунку. Розмір сторони фігури оберіть на власний розсуд (наприклад 150).



Технологія виконання

1. Для спрощення виконання зробіть копію файлу із Завдання №2, що є результатом виконання відповідного завдання та змініть його ім’я на Завдання №3.

2. Відкрийте файл Завдання №3 за допомогою IDLE.

3. Скориставшись досвідом, набутим під час виконання Завдання №2 та знаннями зі шкільного курсу математики (градуска міра одного кута правильного п’ятикутника дорівнює 108 градусів) зробіть у програмі потрібні зміни та доповнення.

4. Збережіть внесені зміни.

5. Запустіть програму на виконання.

6. Перегляньте результати виконання та при потребі зробіть зміни у програмному коді.

Завдання №4

Створіть програму, після виконання якої, на полотні Python Turle Graphics за допомогою модуля Черепашка буде побудовано державний прапор України

(співвідношення довжини та ширини як 3:2, розміри довільні). Збережіть програму у файлі з ім’ям «Завдання 4».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE/

2. Створіть новий файл та збережіть його під ім’ям «Завдання 4»

3. Скориставшись досвідом, набутим під час виконання попередніх завдань розробіть алгоритм малювання прапору та введіть відповідний текст програми

4. Збережіть внесені зміни

5. Запустіть програму на виконання

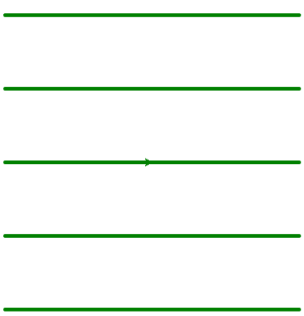
6. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

7. Збережіть зміни та закрийте IDLE [23].

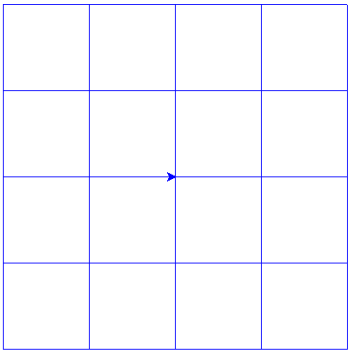
Завдання для самостійного виконання

1. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано горизонтальний відрізок прямої, червоного кольору, довжиною 400 пікселів, не повинно буди подвійних ліній (економія фарби). Після виконання – Черепашка у початковому положенні.

2. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано 5 горизонтальних відрізків прямої зеленого кольору, довжиною 400 пікселів, товщиною 5 пікселів (зразок зображено на рисунку). Повинні виконуватися умови економії «фарби» (довжина шляху який Черепашка проходить з опущеним пером повинна бути мінімальною) та «пального» (довжина шляху який черепашка проходить повинна бути як можна меншою). Після виконання – Черепашка у початковому положенні.



3. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано у центрі полотна квадрат згідно зразка, поданого на рисунку, розміром 400×400 пікселів, товщиною лінії 1 піксель. Повинні виконуватися умови економії «фарби» (довжина шляху який черепашка проходить з опущеним пером повинна бути мінімально) та «пального» (довжина шляху який черепашка проходить повинна бути як можна меншою). Після виконання – Черепашка у початковому положенні.



4. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано зафарбований рівнобедренний трикутник. Після виконання – Черепашка у початковому положенні.

5. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано зафарбований будинок (квадрат під трикутником), без підйому пера при умові однократного переміщення по одній лінії. Після виконання – Черепашка у початковому положенні.

6. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано зафарбованого сніговика. Після виконання – Черепашка у початковому положенні [24].

7. Відтворіть зображення, які попередньо завантажте за посиланням: https://drive.google.com/drive/folders/1TlNGJ6KYQGt01RXgaBlb2TEbyVBqFjTk?usp=sharing.

3.3. Базові елементи в Python

3.3.1. Вбудовані функції

В мові Python поряд з вбудованими типами є вбудовані функції, які містяться в стандартній бібліотеці і доступні без будь-яких додаткових вказівок. Розглянемо функції, що можуть бути застосовані до цілих та дійсних чисел.

abs(X) – повертає абсолютне значення (модуль) числа.

divmod(A, B) – повертає пару чисел (*P, R*), які є цілою частиною P та остачею R при виконанні цілочисельного ділення. Для цілих чисел результат буде таким самим, як і при (*A // B, A % B*). Для дійсних чисел результатом є (*Q, A % B*), де Q зазвичай *math.floor(A / B)*, але може бути і на 1 менше. Незважаючи на це значення за виразом *Q * B + A % B* дуже близьке до A, якщо *A % B* не рівне нулю, то має такий самий знак, як і *B, 0 <= abs(A%B) < abs(B)*.

```
>>> divmod(7,2)
(3, 1)
```

pow(X, Y[, Z]) – повертає X в степені Y за модулем Z (обчислюється більш ефективно, чим *pow(X, Y) % Z)*. Двоаргументна форма *pow(X, Y)* – еквівалент використання оператора піднесення до степеня: *X**Y*. Якщо параметр Z заданий, то X та Y мають бути цілочисельними, окрім того Y має бути невід’ємним.

```
>>> pow(2,3)
8
>>> pow(2,3,3)
2
```

round(number[, ndigits]) – повертає число number, округлене до ndigits знаків після десяткової точки. Проте поведінка round() для дійсних чисел може бути несподіваною, це результат факту, що деякі десяткові дроби не можуть бути представлені точно як дійсні числа.

```
>>> round(2.65, 1)
2.6
>>> round(2.75, 1)
2.8
>>> round(2.85, 1)
2.9
>>> round (2.665, 2)
2.67
>>> round (2.675, 2)
2.67
```

Якщо параметр *ndigits* не заданий, то округлення відбувається до найближчого цілого числа. Проте, якщо дробова частина = 0.5, то виконується «Банківське округлення», тобто округлення до найближчого парного числа.

```
>>> round (2.5, 2)
2
>>> round (3.5, 2)
4
```

max(arg1, arg2, *args, *[, key=func]) – повертає найбільше значення з двох чи більше аргументів. Ця функція має більш широкі можливості, але про них пізніше.

min(arg1, arg2, *args, *[, key=func]) – повертає найменше значення з двох чи більше аргументів. Ця функція має більш широкі можливості, але про них пізніше.

int([object], [osn]) – повертає перетворене значення object до цілого десяткового числа. osn визначає систему числення задання object (osn від 2 до 36 включно). За замовчуванням object=0, osn=10.

```
>>> int(4.9)
4
>>> int('11')
11
>>> int('11',2)
3
```

float([X]) – повертає перетворене X до дійсного числа.

```
>>> float('1.23')
```

```
1.23
>>> float('1e-003')
0.001
>>> float(3)
3.0
```

bin(X) – повертає рядковий запис цілого числа X в двійковій формі.

```
>>> bin(5)
'0b101'
```

hex(X) – повертає рядковий запис цілого числа X в шістнадцятковій формі. Для перетворення дійсного числа використовується метод *float.hex(X)*.

```
>>> hex(255)
'0xff'
>>> float.hex(3.4)
'0x1.b333333333333p+1'
```

oct(X) – повертає рядковий запис цілого числа X у вісімковій формі.

```
>>> oct(12)
'0o14'
```

bool([X]) – повертає приведене значення X до логічного типу (bool), використовуючи стандартну процедуру перевірки істинності.

Повертає логічне значення True або False.

```
>>> bool(1)
True
>>> bool(15)
True
>>> bool(0)
False
```
[12].

3.3.2. Модуль math. Модуль random

Нам вже відомі прості дії з числами та текстом у Python. Але, так як Python – це високорівнева мова програмування, вона має багато додаткових функцій для роботи з числами та текстом, та щоб не навантажувати написану нами програму існують спеціальні набори різноманітних функцій, які називаються модулями.

Додаткові модулі потрібно підключати окремо. Це зроблено, щоб забезпечити швидкість виконання програми.

Отже для того, щоб підключити додатковий модуль до нашої програми потрібно застосувати таку конструкцію:

import назва_модуля

В Python існує велика кількість додаткових модулів для роботи з даними, і якщо у подальшому вашому житті ви плануєте працювати у сфері розробки програмного забезпечення і саме із Python, то рекомендовано знати та уміти працювати із основними з них. Наприклад: існує такий модуль *unittest*, який створений для тестування своїх програм на Python, а ми будемо працювати з модулем *math*, який представляє собою пакет з додатковими функціями для роботи з числами.

А функції підключеного модулю вводяться в код наступним чином: назва_модуля.назва_функції ().

Модуль math

Як ви вже дізналися, модуль math використовується для роботи з числами, та виконання з ними додаткових дій.

Отже, для того щоб підключити даний модуль потрібно лише на початку програмного коду встановити наступну конструкцію коду:

import math

Зверніть увагу! Підключати модулі потрібно саме на початку, бо, якщо ви підключите його вже після виконання функції, яка входить в даний модуль, то вона не буде зрозуміла інтерпретатору і ваша програма не буде виконана.

Що стосується можливостей даного модуля, то в нього входить велика кількість функцій для роботи із числами. Наприклад: math.sqrt(x), яка знаходить квадратний корінь від числа x, або math.fabs(x), яка знаходить модуль від числа x, та багато інших [11].

Розглянемо стандартні константи та основні функції, що надає модуль math (табл. 3.2.9) [24].

Таблиця 3.2.9

Синтаксис	***Призначення***	***Приклади використання***

pi	Повертає число π.	`>>> import math`
		`>>> math.pi`
		`3.141592653589793`

e	Повертає значення константи **e**.	`>>> math.e` `2.718281828459045`

sin()
cos()
tan()
	Стандартні тригонометричні функції (синус, косинус, тангенс). Значення аргументу вказується в радіанах.	`>>> a=math.pi/6`
		`>>> math.sin(a)`
		`0.49999999999999994`
		`>>> math.sin(1.7)`
		`0.9916648104524686`
		`>>> math.cos(a)`
		`0.8660254037844387`

asin()
acos()
atan()
	Обернені тригонометричні функції (арксинус, арккосинус, арктангенс). Значення повертається в радіанах.	`>>> math.asin(0)`
		`0.0`
		`>>> math.asin(0.999)`
		`1.526071239626163`
		`>>> math.acos(-0.999)`
		`3.09686756642106`

degrees()	Перетворює радіани у градуси.	`>>> math.degrees(math.pi/6)` `29.999999999999996`
radians()	Перетворює градуси у радіани.	`>>> math.radians(180.0)` `3.141592653589793`

exp()	Експонента.	`>>> import math`
		`>>> math.exp(0)`
		`1.0`
		`>>> math.exp(2)`
		`7.38905609893065`

log()	Логарифм.	`>>> math.log(8, 2)`
		`3.0`
		`>>> math.log(100, 10)`
		`2.0`

sqrt()	Квадратний корінь	`>>> math.sqrt(49)`
		`7.0`
		`>>> math.sqrt(100)`
		`10.0`

ceil()	Значення, округлене до найближчого більшого цілого.	`>>> math.ceil(6.49)` `7`
floor()	Значення, округлене до найближчого меншого цілого.	`>>> math.floor(7.77)` `7`
pow(<Число>, <Степінь>)	Підносить <Число> до <Степені>.	`>>> math.pow(3, 4)` `81.0`
fabs()	Абсолютне значення.	`>>> math.fabs(-14.5)` `14.5`

fmod()	Остача від ділення.	`>>> math.fmod(25, 5)`
		`0.0`
		`>>> math.fmod(25, 7)`
		`4.0`

factorial()	Факторіал числа.	`>>> math.factorial(3)` `6`
modf(x)	Повертає дробову і цілу частину x. Обидва числа мають той же знак, що й x.	`>>> math.modf(4.07)` `(0.07000000000000028, 4.0)`

Модуль random

Більшість програм роблять одне і те ж при кожному виконанні, тому говорять, що такі програми визначені. Визначеність хороша річ до тих пір, поки ми вважаємо, що одні й ті ж обчислення повинні давати один і той же результат. Проте, в деяких програмах від комп'ютера потрібно непередбачуваність. Типовим прикладом є ігри, але є маса інших застосувань: зокрема, моделювання фізичних процесів або статистичні експерименти.

Змусити програму бути дійсно непередбачуваною завдання не таке просте, але є способи змусити її здаватися непередбачуваною. Одним з таких способів є генерування випадкових чисел і використання їх у програмі.

У Python є вбудований модуль, який дозволяє генерувати псевдовипадкові числа. З математичної точки зору, вони не істинно випадкові.

Модуль random дозволяє генерувати випадкові числа [31]. Перш ніж використовувати модуль, необхідно підключити його за допомогою інструкції:

import random

Основні функції:

- **random()** – повертає псевдовипадкове число від 0.0 до 1.0:

```
>>> import random
>>> random.random()
0.5496035697985201
>>> random.random()
0.03130983927165354
```

- **seed ([<Параметр>] [, version=2])** – налаштовує генератор випадкових чисел на нову послідовність. За замовчуванням використовується системний час. Якщо значення параметра буде однаковим, то генерується однакове число:

```
>>> random.seed(7)
>>> random.random()
0.32383276483316237
>>> random.random()
0.15084917392450192
>>> random.seed(7)
>>> random.random()
0.32383276483316237
```

- **uniform(<Початкове значення>, <Кінцеве значення>)** – повертає псевдовипадкове дійсне число в діапазоні від <Початкове значення> до <Кінцеве значення>:

```
>>> import random
>>> random.uniform(0, 7)
2.998147608364766
>>> random.uniform(0, 7)
1.264932072954918
```

- **randint(<Початкове значення>, <Кінцеве значення>)** – повертає псевдовипадкове ціле число в діапазоні від <Початкове значення>, до <Кінцеве значення>:

```
>>> random.randint(0, 7)
7
>>> random.randint(0, 7)
4
>>> random.randint(0, 100)
50
```

- **randrange([<Початкове значення>,] <Кінцеве значення>[, <Крок>])** – повертає випадковий елемент з числової послідовності. Параметри аналогічні параметрам функції range(). Саме зі списку, що повертається функцією range(), і вибирається випадковий елемент:

```
>>> random.randrange(10)
9
>>> random.randrange(0, 7)
2
>>> random.randrange(0, 15, 3)
12
```

- **choice(<Послідовність>)** – повертає випадковий елемент з будь-якої послідовності (рядка, списку, кортежу):

```
>>> random.choice("рядок")
'р'
>>> random.choice("рядок")
'я'
```

- **shuffle(<Список>[, <Число від 0.0 до 1.0>])** – перемішує елементи списку випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, то використовується значення, яке повертається функцією random(). Приклад:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> random.shuffle(arr)
>>> arr
[5, 1, 6, 4, 2, 3, 7]
```

- **sample(<Послідовність>, <Кількість елементів>)** – по-вертає список із зазначеної кількості елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. Як послідовності можна вказати будь-які об’єкти, що підтримують ітерації [24]. Приклади:

```
>>> random.sample("рядок", 2)
['д', 'р']
>>> random.sample(range(100), 7)
[62, 53, 23, 32, 51, 82, 41]
```

Контрольні запитання

1. Які числові типи підтримує Python 3?

2. В мові Python якого типу буде результат операції у якій беруть участь ціле і дійсне числа?

3. Як на мові Python записується математичний оператор «Ціла частина від ділення»?

4. Який оператор на мові Python записується наступним чином: a % b?

5. Як на мові Python записується математичний оператор «Піднесення до степеня»?

6. Перерахуйте відомі вам оператори присвоювання у мові програмування Python? Як вони позначаються та виконуються?

7. Який пріоритет математичних операторів у Python? За допомогою чого його можна змінити?

8. Назвіть відомі вам вбудовані функції для роботи з числами.

9. Яке призначення модуля math?

10. Що дозволяє робити модуль random?

Практичні завдання

Завдання №1

Складіть програму для обчислення площі трапеції ($S = \frac{a+b}{2}h$) за відомими основами (a, b) та висотою (h) та збережіть її з назвою «Завдання №1». Виконайте її за допомогою інтегрованого середовища IDLE.

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.

2. Створіть новий файл та збережіть його під ім'ям «Завдання 1».

3. Введіть текст програми:

```
print("Введіть основи (a і b) та висоту (h) трапеції.")
a=float(input('a= '))
b=float(input('b= '))
h=float(input('h= '))
print('S= ', (a+b)/2*h)
input('Для завершення роботи натисніть клавішу <Enter>.')
```

4. Збережіть внесені зміни.

5. Запустіть програму на виконання.

6. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконання програму.

7. Збережіть зміни та закрийте IDLE.

Завдання №2

Створіть програму для обчислення значень функції $f(x) = 5x^4 - 3x^2 + 7x - 15$ та збережіть її у файлі «Завждання №2». Виконайте програму за допомого інтегрованого середовища розробки IDLE.

Технологія виконання

1. Відкрийте інтегроване середовище розробки IDLE.

2. Створіть новий файл та збережіть його під ім'ям «Завдання №2».

3. Введіть текст програми. Він може бути таким:

```
print('Введіть значення аргумента функції.')
x=float(input('x = '))
print('f(', x, ') = ', 5*x**4-3*x**2+7*x-15)
input("Для завершення натисніть <Enter>.")
```

4. Збережіть внесені зміни.

5. Запустіть програму на виконання.

6. Перегляньте результати виконання програми та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

7. Збережіть зміни та закрийте IDLE.

Завдання №3

Із пунктів А і В, розташованих одне від одного на віддалені d км, назустріч один одному одночасно відправляються два поїзди; швидкість першого V_1 км/год, швидкість другого V_2 км/год. В цей же час із пункту А вилітає надшвидкісна муха із швидкістю V км/год і летить назустріч поїзду з пункту В. Зустрівшись з ним, вона летить до поїзда із пункту А, і т. д. до тих пір, поки поїзди не зустрінуться. Визначити загальну відстань, яку пролетить муха.

Технологія виконання

1. Відкрийте інтегроване середовище розробки IDLE.

2. Створіть новий файл та збережіть його під ім’ям «Завдання №3».

3. Якщо у вас виникли труднощі при створенні алгоритму для розв’язування задачі ви можете скористатись підказкою.

Підказка

Загальний шлях мухи залежить від часу її польоту, який дорівнює часу руху поїздів до зустрічі. А цей час дорівнює відношенню відстані між містами до сумарної швидкостіі поїздів (швидкості зближеня).

4. Текст програми може бути такий:

```
d=float(input('d = '))
V1=float(input('V1 = '))
V2=float(input('V2 = '))
Vm=float(input('Vm = '))
print('S = ', d/(V1+V2)*Vm)
input("Для завершення натисніть <Enter>.")
```

5. Збережіть внесені зміни.

6. Запустіть програму на виконання.

7. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

Завдання №4

Складіть програму для обчислення значень функції $f(x) = \sin(x) + cos^2(x)$.

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.

2. Створіть новий файл та збережіть його під ім’ям «Завдання №4».

Підказка

Для використання тригонометричних функцій потрібно приєднати модуль math.

3. Створіть алгоритм та введіть текст програми.

```
import math
print('Введіть значення аргумента функції.')
x=float(input('x = '))
print('f(', x, ') = ', math.sin(x)+(math.cos(x))**2)
input("Для завершення натисніть <Enter>.")
```

4. Збережіть внесені зміни.

5. Запустіть програму на виконання.

6. Перегляньте результати виконання та при потребі зробіть необхідні корективи.

Завдання №5

Створіть програму, для визначення цифр тризначного натурального числа.

Технологія виконання

1. Створіть у середовищі IDLE новий файл та збережіть його під ім’ям «Завдання №5».

2. Для визначення цифри одиниць цифри можна скористатись операцією Остача від ділення (%), а для визначення цифри десятків та сотень операціями Остача від ділення (%) та Цілаа частина вділення (//).

3. Тоді програма може бути такою:

```
n=int(input('Введіть число: '))
print('Цифра одиниць: ', n%10)
print('Цифра десятків: ', (n//10)%10)
print('Цифра сотень: ', (n//100)%10)
input()
```

4. Запустіть програму на виконання.

5. Перегляньте результати виконання та при потребі зробіть правки [24].

Завдання для самостійного виконання

1. Складіть програму для обчислення площі паралелограма ($S = ah$) за відомою основою (a) та висотою (h) та збережіть її у файлі. Виконайте її за допомогою інтегрованого середовища IDLE.

2. Скласти програму обміну значеннями між змінними A і B, не застосовуючи третю змінну та оператор обміну. Значення змінних – дійсні числа.

3. Створіть та збережіть програму, після виконання якої, на полотні за допомогою модуля Черепашка буде побудовано вертикальний відрізок прямої, червоного кольору, довжиною **b**, значення якої повинен ввести користувач під час виконання програми. Повинна виконуватися умова економії «фарби» (довжина шляху який Черепашка проходить з опущеним пером повинна бути мінімальною). Після виконання – Черепашка у початковому положенні, перо підняте.

4. Створіть програму, для визначення цифр двозначного натурального числа.

5. Створіть програму, для визначення суми та добутку цифр двозначного натурального числа.

6. Створіть програму, для визначення цифр чотиризначного натурального числа.

7. Створіть програму, для визначення суми та добутку цифр тризначного натурального числа.

8. Створіть програму, для визначення числа, що утвориться у результаті перестановки одиниць та сотень даного чотиризначного натурального числа.

9. Степан вирішив пригостити однокласників шоколадками. Шоколадка коштувала ***N*** грн. З першого листопада вартість шоколадки збільшилась рівно на ***P*** відсотків. Визначте, скільки шоколадок зможе купити Степан на ***S*** грн після подорожчання.

10. Михайлик любив малювати трикутники, але він це робив у незвичний спосіб. Спочатку малював довільний трикутник, потім кожну сторону ділив на **n** рівних частин і проводив через точки поділу прямі, паралельні сторонам трикутника. У результаті виходить декілька рівних між собою трикутників. Допоможіть Михайлику знайти найбільшу кількість рівних трикутників у його фінальному рисунку.

11. Мама попросила Васю полити всі молоді деревця у саду. Вася знає, що поки дерева маленькі, їх потрібно дуже добре поливати. А ось скільки поливати – невідомо. Але Вася – дуже розумний хлопчик. Він уважно прочитав весь підручник ботаніки для середньої школи і вияснив, що полив прямо пропорційний кількості листочків на дереві. Для гарного росту дерев достатньо виливати під дерево щоденно по одному літру води для кожного листка. На щастя Васі виявилось, що листки на деревах ростуть ярусами, причому на верхньому ярусі два листка, на другому – чотири, на наступному – шість, і так далі, на кожному наступному ярусі на два листки більше у порівнянні з попереднім. А на самій верхівці росте ще один листочок. Хитрий Вася послав молодшу сестричку Машеньку підрахувати кількість ярусів на кожному дереві, а Вас просить написати програму, яка для кожного дерева обрахує кількість літрів води для його поливу.

12. Вартість пляшки води, враховуючи вартість порожньої пляшки, становить 1 грн 20 коп., а вартість порожньої пляшки 20 коп. Скільки пляшок води можна випити на n грн, враховуючи, що порожні пляшки можна здавати, і на одержані гроші купувати нові пляшки води.

13. «Гарним» будемо вважати число, що складається лише з непарних цифр. Наприклад число 157953 гарне, а число 2452117 ні. Необхідно з'ясувати, скільки існує n – значних гарних чисел.

14. Пиріжок у шкільній їдальні коштує a гривень та b копійок. Знайдіть скільки гривень та копійок заплатить Петрик за n пиріжків.

15. Учні 10-Б класу, на осінні канікули, вирішили поїхати на екскурсію до столиці. Знаючи кількість хлопчиків n та дівчаток m, визначити скільки потрібно замовити кімнат в готелі, в якому є кімнати на k місць кожна, за умови, що хлопчиків та дівчаток поселяти разом заборонено.

16. Знайти суму цифр даного двоцифрового числа.

17. У заданому трицифровому натуральному числі поміняти першу та останню цифри місцями.

18. Знайдіть квадрат суми цифр чотирицифрового натурального числа.

19. Знайти добуток цифр п'ятицифрового числа ***n***, які стоять на непарних розрядах.

20. Задано натуральне число ***N***. Напишіть програму, яка знаходить кількість натуральних чисел, що не перевищують ***N*** і не діляться на жодне з чисел 2, 3, 5 [24].

3.3.3. Умовні оператори. Оператор розгалуження if … else

Умовний оператор (або, як його ще іноді називають, умовна інструкція) залежно від істинності чи хибності певної умови дозволяє виконувати різні блоки програмного коду. Загальна схема, відповідно до якої функціонує умовний оператор, виглядає так:

- Перевіряється деяка умова – зазвичай це вираз, значення якого може інтерпретуватися як істинне (значення True) або хибне (значення False).
- Якщо умова (вираз) інтерпретується як істинна, виконується виділена спеціальним чином послідовність команд.
- Якщо умова інтерпретується як хибна, виконується інша (але теж виділена спеціальним чином) послідовність команд.

- Після того, як одну або іншу послідовність команд умовного оператора виконано, керування передається тій команді, яка знаходиться після команди виклику умовного оператора.

Фактично, є два набори команд, а рішення про те, який із наборів команд виконати, приймається за результатами перевірки умови-виразу. Таким чином, у програмі створюється «точка розгалуження».

У мові Python умовний оператор (оператор розгалудження) реалізується у вигляді програмного блоку з наступним шаблоном (жирним шрифтом виділено ключові елементи):

if умова:
  команди_1
else:
  команди_2

Після ключового слова *if* указується умова – вираз логічного типу (або який допускає інтерпретацію в термінах *True* і *False*). Після умови ставиться двокрапка (тобто :). Далі йде блок команд, які виконуються, якщо умова істинна (у шаблоні це команди_1). Блок команд виділяється за допомогою відступів (це стандартний спосіб виділення блоків команд у мові Python). У принципі, кількість відступів може бути довільною – головне, щоб для кожної команди блоку робилася одна й та сама кількість відступів. Але загальноприйнятим є правило робити 4 відступи (4 пробіли).

Після закінчення блоку команд (виконуваних за істинної умови) іде ключове слово *else* (і двокрапка :). Ключове слово *else* повинно бути на тому ж рівні (така ж кількість відступів або пробілів), що й ключове слово *if*. Далі – ще один блок команд (у шаблоні позначено як команди_2), які виконуються, якщо умова хибна [4].

Розглянемо роботу оператора розгалуження на прикладі простої програми:

```
a=int(input('a = '))
if a>7:
    print('7')
elif a>3:
    print('6')
elif a>1:
    print('5')
else:
    print('0')
input('Для завершення натисніть <Enter>.')
```

Результати роботи програми для різних значень *a* будуть такими [24]:

```
 RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.py
a = 8
7
Для завершення натисніть <Enter>.
>>>
 RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.py
a = 4
6
Для завершення натисніть <Enter>.
>>>
 RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.py
a = 2
5
Для завершення натисніть <Enter>.
>>>
 RESTART: D:\OneDrive\Робоча документація\rik 2017-2018\
Наукова робота\Посібники\Python для інф\Матеріали\Програ
ми\Приклади\Пункт 1-13-2\Ком-розгал.py
a = 1
0
```

Схема виконання умовного оператора [4] ілюструється на рис. 3.3.3.1.

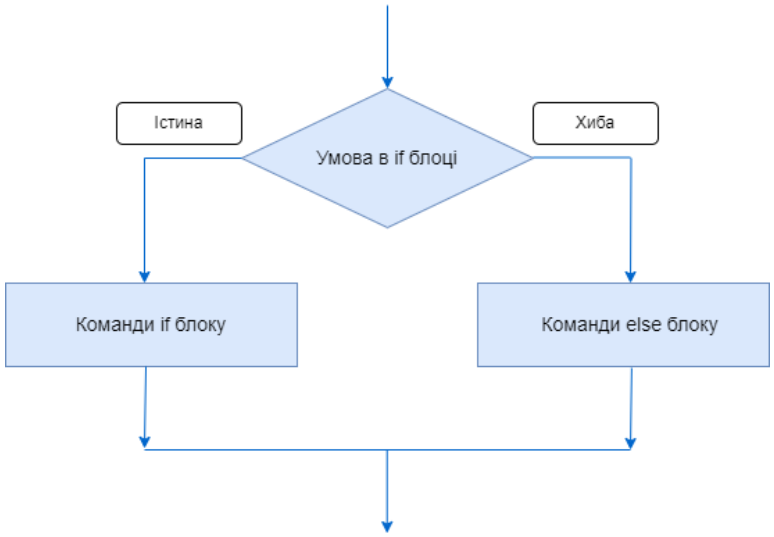


Рисунок 3.3.3.1. Схема виконання умовного оператора

Контрольні запитання

1. Які операції порівняння використовуються у мові Python 3? Який їх синтаксис?

2. Навіщо використовуються логічні операції? Перерахуйте їх.

3. Який синтаксис та правила виконання оператора розгалуження if...else?

Практичні завдання

Завдання №1

Створіть програму, яка перевіряє, чи є введене користувачем натуральне число парним чи ні. Після перевірки повинно вивестися відповідне повідомлення. Збережіть у файлі з назвою «Завдання №1». Виконайте її за допомогою інтегрованого середовища розробки IDLE.

Технологія виконання

1. Відкрийте інтегроване середовище IDLE

2. Створіть новий файл та збережіть його під ім’ям «Завдання №1»

3. При створенні програми використаємо команди введення, виведення та розгалуження:

```
x=int(input("Введіть натуральне число: "))
if x%2==0:
    print(x, '- парне число')
else:
    print(x, '- непарне число')
input()
```

4. Збережіть програму та виконайте її для різних наборів даних

5. Перегляньте результати виконання та при потребі зробіть корективи

Зауваження

Якщо блок складається з однієї інструкції, то цю інструкцію можна розмістити на одному рядку із ключовим словом if та логічним виразом:

```
x=int(input("Введіть натуральне число: "))
if x%2==0: print(x, '- парне число.')
else: print(x, '- непарне число.')
input()
```

Завдання №2

Створіть програму для розв'язування квадратного рівняння $ax^2 + bx + c = 0$. Збережіть її у файлі з назвою «Завдання №2». Протестуйте її за допомогою системи тестів наведеної у таблиці.

Номер теста	**Випадок, що перевіряється**	**Коефіцієнти**			**Результати**
		a	***b***	***c***	
1	*d>0*	1	1	-2	$x_1 = 1, x_2 = -2$
2	*d=0*	1	2	1	Корені дорівнюють: $x_1 = -1, x_2 = -1$
3	*d<0*	2	1	2	Не має дійсних коренів
4	*a=0, b=0, c=0*	0	0	0	Всі коефіцієнти дорівнює нулю. *x* – довільне число
5	*a=0, b=0, c<>0*	0	0	2	Невірне рішення
6	*a=0, b<>0*	0	2	1	Лінійне рівняння. Один корінь: $x = -0{,}5$
7	*a<>0, b<>0, c=0*	2	1	0	$x_1 = 0, x_2 = -0{,}5$

Технологія виконання

1. За допомогою інтегрованого середовища IDLE створіть новий файл та збережіть його під ім'ям «Завдання №2».

2. Перед створенням програми для розв'язування квадратного рівняння потрібного спочатку проаналізувати різні види рівнянь, що ми отримаємо у залежності від значень коефіцієнтів *a*, *b* та *c*. Це завдання значно спрощується завдяки наведеній системі тестів. Результати аналізу відповідно необхідно врахувати при створенні алгоритму та написанні програми.

3. Введіть програму, збережіть її та виконайте для різних наборів даних, наведених у системі тестів.

4. Виконайте програму для кількох власних наборів даних.

Підказка

Для обчислення $\sqrt{D}$ потрібно приєднати модуль math

Завдання №3

Створіть програму для виведення на екран більшого з трьох введених користувачем чисел. Функції (такі, наприклад, як *max()*, *min()* тощо) використовувати забороняється. Ім'я файлу для збереження «Завдання №3».

Технологія виконання

1. Відкрийте інтегроване середовище IDLE.

2. Створіть новий файл та збережіть його під ім'ям «Завдання №3».

3. Використовуючи оператори розгалуження, порівняйте числа між собою та виведіть потрібний результат на екран.

```
a1=float(input('Перше число: '))
a2=float(input('Друге число: '))
a3=float(input('Третє число: '))
if a1>=a2 and a1>=a3:
    print('Найбільше число', a1)
elif a2>=a1 and a2>=a3:
    print('Найбільше число', a2)
else:
    print('Найбільше число', a3)

input()
```

4. Введіть код та збережіть програму. Виконайте її для різних наборів даних.

5. Перегляньте результати виконання програми та при потребі змініть код програми.

Завдання №4

Створіть програму для обчислення значень функції.

$$y = \begin{cases} x^2 + 1, \text{якщо } x > 3; \\ x - 4, \text{якщо } x \leq 3. \end{cases}$$

Технологія виконання

1. За допомогою IDLE створіть та збережіть новий файл під ім’ям «Завдання №4».

2. Досвід здобутий вами під час виконання Завдань 1-3 дозволить вам з легкістю виконати це завдання.

```
x=float(input('x='))
if x>3:
    print('y=', x**2+1)
else:
    print('y=', x-4)

input("Для завершення натисніть <Enter>.")
```

3. Виконайте програму для кількох значень аргументу.

4. Перегляньте результати виконання та при потребі змініть код [24].

Завдання для самостійного виконання

1. Складіть програму, яка перевіряє, чи ділиться введене користувачем число на 7 чи ні. Після перевірки повинно вивестися відповідне повідомлення.

2. Створіть програму для розв’язування лінійного рівняння $ax + b = 0$.

3. Створіть програму для виведення на екран більшого з двох введених користувачем чисел.

4. Вивести на екран номер чверті, якій належить точка з координатами *(x, y)* за умови, що *x* та *y* відмінні від 0.

5. Дано натуральне двозначне число. Написати програму для визначення, чи є сума його цифр двозначним числом.

6. Дано натуральне двозначне число. Написати програму для визначення, чи є більшою цифра десятків за цифру одиниць, чи вони рівні.

7. Дано натуральне тризначне число. Написати програму для визначення чи є воно паліндромом («перевертнем»), іншими словами числом, десятковий запис якого читається однаково зліва направо і справа наліво.

8. Створіть програму для обчислення значень функції

$$y = \begin{cases} x - 7, \text{якщо } x < 0, \\ 5, \text{якщо } 0 \leq x \leq 5, \\ x^2 + 5, \text{якщо } x > 5 \end{cases}$$

9. Дано три різних цілих числа, знайти середнє з них. Середнім назвемо число, яке більше найменшого з даних чисел, але менше максимального.

10. Складіть програму знаходження добутку двох найбільших з трьох введених з клавіатури чисел.

11. Дано тризначне натуральне число. Написати програму для визначення чи входить до нього цифра 4.

12. Дано двозначне натуральне число. Написати програму для визначення чи входять до нього цифри 5 і 7.

13. Складіть програму, яка з трьох введених з клавіатури чисел підносить до квадрату додатні, а від'ємні залишає без зміни.

14. Степан влітку відпочиває у бабусі в селі. Особливо йому подобається купатись на сільському озері. Посередині озера плаває пліт, який має форму прямокутника. Сторони плота спрямовані уздовж паралелей і меридіанів. Введемо систему координат, в якій вісь *OX* направлена на схід, а вісь *OY* – на північ. Нехай південно-західний кут плоту має координати (x_1, y_1), північно-східний кут – (x_2, y_2). Степан знаходиться в точці з координатами (x, y). Визначте, до якої сторони плоту (північної, південної, західної чи східної) або до будь-якого кута плоту (північно-західному, північно-східному, південно-західному, південно-східному) Степану потрібно плисти, щоб якомога швидше дістатись до плоту.

Формат вхідних даних:

Дано шість чисел в наступному порядку: x_1, y_1 (координати південно-західного кута плоту), x_2, y_2 (координати північно-східного кута плоту), x, y (координати Степана). Всі числа цілі і по модулю не перевершують 100.

Гарантується, що $x_1 < x_2$, $y_1 < y_2$, $x \neq x_1$, $x \neq x_2$, $y \neq y_1$, $y \neq y_2$, координати Степана знаходяться поза плотом.

Формат вихідних даних:

Якщо Степану слід пливти до північної сторони плоту, програма повинна вивести символ «*N*», до південної - символ «*S*», до західної - символ «*W*», до

східної – символ «*E*». Якщо Степану слід пливти до кута плоту, потрібно вивести один з наступних рядків: «*NW*», «*NE*», «*SW*», «*SE*» [24].

15. На сковорідку одночасно можна покласти k котлет. Кожну котлету потрібно з кожного боку обсмажувати m хвилин безперервно. За який найменший час вдасться підсмажити з обох сторін n котлет?

Формат вхідних даних:

Програма отримує на вхід три числа: k, m, n.

Формат вихідних даних:

Програма повинна вивести одне число: найменшу кількість хвилин.

16. Уздовж прямої викладені три сірники. Необхідно перекласти один із них так, щоб при підпалюванні будь-якого сірника згорали всі три. Для того щоб вогонь переходив з одного сірника на інший, необхідно щоб ці сірники дотикалися (хоча б кінцями).

Потрібно написати програму, яка визначить, яку з трьох сірників необхідно перемістити.

Формат вхідних даних:

Вводяться шість цілих чисел: $l_1, r_1, l_2, r_2, l_3, r_3$ - координати першого, другого і третього сірника відповідно ($0 \le l_i < r_i \le 100$). Кожен сірник описується координатами лівого і правого кінця по горизонтальній осі OX.

Формат вихідних даних:

Виведіть номер шуканого сірника. Якщо можливих відповідей декілька, то виведіть найменший з них. У разі, коли немає необхідності переміщати будь-який сірник, виведіть 0. Якщо ж необхідного результату досягти неможливо, то виведіть -1.

17. Дано три сторони трикутника a, b, c. Визначте тип трикутника із заданими сторонами. Виведіть одне з чотирьох слів: rectangular для прямокутного трикутника, acute для гострокутного трикутника, obtuse для тупокутного трикутника або impossible, якщо трикутника з такими сторонами не існує.

Формат вхідних даних:

Вводяться три цілих числа.

Формат вихідних даних:

Виведіть відповідь до задачі. [35].

18. Вивести кількість днів в n–му місяці m–го року, враховуючи, що рік є високосним, якщо він кратний 4, але не кратний 100, або ж кратний 400.

Формат вхідних даних: Значення n та m ($1 \le n \le 12$, $1 \le m \le 2100$).

Формат вихідних даних: Знайдена кількість днів.

19. Дано натуральне число N ($1 \le N \le 100$), яке визначає вік людини. Виведіть це число з відповідним доданком rik, roki, rokiv. Наприклад, 1 rik, 12 rokiv, 94 roki.

Формат вхідних даних: Єдине число - вік людини N ($1 \le N \le 100$).

Формат вихідних даних: Вивести вік з відповідною назвою [15].

20. Рівняння для п’ятикласників представляє собою рядок довжиною 5 символів. Другий символ рядка є або знаком «+» (плюс) або «-» (мінус), четвертий символ – знак «=» (дорівнює). З першого, третього і п’ятого символів рівно два є цифрами з діапазону від 0 до 9, і один – буквою *x*, яка позначає невідоме.

Потрібно написати програму, яка дозволяє вирішити дане рівняння відносно *x*.

Формат вхідних даних: Один рядок, в якому записано рівняння.

Формат вихідних даних: Виведіть ціле число – значення *x* [5].

3.4. Цикли в Python

3.4.1. Цикли

Досить часто і в житті, і при написанні програм існує необхідність повторення деякої дії певної кількості раз. Тобто при написанні програм може знадобитися певна конструкція, за якою можна буде організувати повторне виконання операторів. Таку конструкцію називають *конструкцією повторення* або *циклом*. А кожну повторену дію – кроком циклу або ітерацією. Отже, можна

зазначити, що *цикл* у програмуванні – це повторюване виконання одних і тих самих простих або складених операторів.

Всі цикли складаються з заголовку та тіла циклу. Заголовок циклу відповідає за налагодження циклу, тобто умову повторення циклу. Тіло ж циклу відповідає за самі дії, які мають повторно виконуватися. Так наприклад, уявімо собі першокласника, який дуже полюбляє морозиво. Йому мама видала певну суму грошей на морозиво. Зрозуміло, він побіг його купувати, але згадав, що множити та ділити він не вміє. Як же йому вирішити цю проблему? Спочатку він перевірить, чи вистачить йому грошей на купівлю пачки морозива. Якщо так, то він її купить і знову погляне на залишок грошей. В цьому прикладі можна виділити заголовок циклу – поки грошей достатньо, і тіло циклу – купівля морозива [12].

В програмуванні розрізняють такі види циклів:

1. Цикл з передумовою, що виконує дії, поки умова істинна (цикл while).

2. Цикл з після-умовою, який спочатку виконує команди, а потім перевіряє умову (do…while).

3. Цикл з лічильником, який виконується задану кількість разів (цикл for).

4. Сумісний цикл, який виконує команди для кожного елементу із відповідного заданого набору значень (for…which).

Python використовує лише цикл з передумовою (while) та цикл for, які є по суті, циклом з лічильником та сумісним циклом [6].

3.4.2. Цикл for

Припустимо, потрібно вивести всі числа від 1 до 70 по одному у рядку. Звичайним способом довелося б писати 70 рядків коду:

```
print(1)
print(2)
…
print(70)
```

За допомогою циклів ту саму дію можна виконати одним рядком коду:

```
>>> for x in range(1, 71): print(x)
```

Іншими словами, цикли дозволяють виконати одні й ті ж інструкції багаторазово.

У мові Python використовуються два цикли: for і while. Цикл for найчастіше застосовується для перебору елементів послідовності. Він має такий формат:

for <Поточний елемент> in <Послідовність>:

<Інструкції всередині циклу>

[else:

<Блок, що виконується, якщо не використовувався оператор break>]

Тут є такі конструкції:

- <Послідовність> – об'єкт, що підтримує механізм ітерації. Наприклад, рядок, список, кортеж, словник та ін.;
- <Поточний елемент> – на кожній ітерації через цей параметр є доступним поточний елемент послідовності або ключ словника;
- <Інструкції всередині циклу> – блок, який буде багаторазово виконуватися;
- якщо всередині циклу не використовувався оператор break, то після завершення виконання циклу буде виконано блок в інструкції else. Даний блок не є обов'язковим [24].

Приклади використання циклу for

Створимо програму для обчислення суми квадратів n перших натуральних чисел ($s = 1 + 2^2 + 3^2 + \cdots + n^2$).

Алгоритми знаходження сум подібного типу мають схожу структуру:

- вводимо значення n;
- присвоюємо змінній, у якій буде зберігатися сума, початкове значення;
- використовуємо інструкцію циклу для обчислення суми;
- виводимо результат.

Реалізація алгоритму цієї задачі на мові Python може бути, наприклад, такою:

```
n=int(input("n="))
s=0
for i in range(1, n+1):
    s=s+i**2
print("S=", s, sep="")
input()
```

Для розуміння виконання циклу розглянемо як будуть змінюватися значення змінних при виконанні програми для $n = 4$.

n	4	4	4	4	4
i		1	2	3	4
s	0	1	5	14	30

А тепер створимо програму для обчислення суми квадратів перших парних натуральних чисел, що не перевершують задане натуральне число $n (s = 2^2 + 4^2 + 6^2 + 8^2 + \cdots + k^2,$ де $k \leq n)$.

Алгоритм для розв'язування цієї задачі буде майже аналогічним, єдине, що потрібно змінити та додати це значення параметрів функції *range()*.

```
n=int(input("n="))
s=0
for i in range(2, n+1, 2):
    s=s+i**2
print("S=", s, sep="")
input()
```

3.4.3. Функція range()

Функція range()

Функція range() має такий вигляд: range ([<Початкове значення>,] <Кінцеве значення>[, <Крок>]). Якщо <Початкове значення> не вказано, то за замовчуванням використовується значення 0. Зауважимо, що кінцеве значення не входить до значень які повертаються. Якщо параметр <Крок> не вказано, то використовується значення.

Розглянемо окремі випадки:

- range(n) – шкала чисел від 0 до n-1;
- range(k, n) – шкала чисел від k до n-1;

• range(k, n, m) – шкала чисел від k до n-1 з кроком m (причому m може бути від'ємним [24].

Контрольні запитання

1. Який синтаксис має функція range?
2. Коли доцільно використовувати цикл for?
3. Який синтаксис циклу for та правила виконання?

Практичні завдання

Завдання №1

Створіть програму для обчислення факторіала. (Факторіал натурального числа n – добуток натуральних чисел від одиниці до n – включно, позначається n!). Збережіть її у файлі «Завдання №1». Обчисліть за допомогою програми факторіали кількох натуральних чисел.

Технологія виконання

1. За допомогою інтегрованого середовища IDLE створіть новий файл та збережіть його під назвою «Завдання №1».

2. Спочатку розглянемо алгоритм розв'язування задачі:

- Ввести значення n;
- Обчислити n;
- Вивести результат.

3. Щоб ввести n використовуємо команду `n=int(input("n="))`.

4. Для збереження значення n! застосуємо змінну f, а для його обчислення використаємо цикл for.

5. Щоб вивести результат використовуємо команду print().

6. У результаті отримуємо:

```
n=int(input("n="))
f=1
for i in range(1, n+1):
    f=f*i
print(n, "!=", f, sep="")
input()
```

7. Збережіть та виконайте програму для кількох значень n.

8. Перегляньте результати виконання та при потребі зробіть у програмі необхідні корективи.

Завдання №2

Дано натуральні числа n, k (n<k, k<9999). З чисел від n до k вибрати ті, запис яких містить рівно три однакових цифри. Наприклад, числа 5855, 5777, 0004, 0200 містять рівно три однакових цифри. Файл зберегти з назвою «Завдання №2».

Технологія виконання

Якщо дане число містить рівно три однакових цифри, то тільки одна з цифр відрізняється від інших, тобто можливі чотири випадки, наведені у таблиці:

	1	2	3	4	
1	*x*	*X*	*x*		Перша умова
2	*x*	*X*		*x*	Друга умова
3	*x*		*x*	*x*	Третя умова
4		*X*	*x*	*x*	Четверта умова

Нехай у якості *n* і *k* введені числа 3532 і 3540. У змінних a_1, a_2, a_3, a_4 зберігаємо значення цифр поточного числа *i.*

i	a_1	a_2	a_3	a_4	
3532	3	5	3	2	False
3533	3	5	3	3	True
3534	3	5	3	4	False
3535	3	5	3	5	False
3536	3	5	3	6	False
3537	3	5	3	7	False
3538	3	5	3	8	False
3539	3	5	3	9	False

3540	3	5	4	0	False

Переходимо до створення програми

1. Створіть новий файл та збережіть його під ім'ям «Завдання №2»

2. Зважаючи на проведений аналіз задачі алгоритм її розв'язання може містити такі частини:

- Введення значень *n* та *k*.
- За допомогою циклу *for* здійснюється перебір чисел на відповідність умові задачі (використовується команда розгалуження) і якщо число задовільняє умови то здійснюється його виведення.

3. Введіть програму, збережіть та виконайте її для кількох значень *n*.

4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму. У якості підказки можете орієнтуватись на даний код.

```
n = int(input("Введіть натуральне число n "))
k = int(input("Введіть натуральне число k<10000 "))

for i in range(n, k+1):
    m = i
    m4 = m%10  # Четверта цифра
    m = m//10
    m3 = m%10  # Третя цифра
    m = m//10
    m2 = m%10  # Друга цифра
    m1 = m//10 # Перша цифра
    if (m1==m2==m3):
        print(m1,m2,m3,m4,sep='')
    elif (m1==m2==m4):
        print(m1,m2,m3,m4,sep='')
    elif (m1==m3==m4):
        print(m1,m2,m3,m4,sep='')
    elif (m2==m3==m4):
        print(m1,m2,m3,m4,sep='')
```

Завдання №3

Степан збирає речі у відпустку. З собою в літак він може взяти ручну поклажу і багаж. Для ручної поклажі у нього є рюкзак, а для багажу – здоровенна валіза. За правилами перевезення маса ручної поклажі не повинна перевищувати *S* кг, а багаж може бути будь-якої маси (за наднормативний багаж Степан

готовий доплатити). Зрозуміло, найбільш цінні речі – ноутбук, фотоапарат, документи і т. д. – Степан хоче покласти в ручну поклажу. Степан розклав усі свої речі в порядку зменшення їх цінності і починає складати найбільш цінні речі в рюкзак. Він діє в такий спосіб – бере найціннішій предмет, і якщо його маса не перевищує *S*, то кладе його в рюкзак, інакше кладе його до валізи. Потім він бере наступний за цінністю предмет, якщо його можна покласти в рюкзак, тобто якщо його маса разом з масою вже покладених в рюкзак речей не перевищує *S*, то кладе його в рюкзак, інакше до валізи, і таким же чином процес триває для всіх предметів в порядку спадання їх цінності.

Визначте вагу рюкзака і валізи після того, як Степан складе всі речі.

Формат вхідних даних:

Перший рядок вхідних даних містить число S – максимально дозволена вага рюкзака. У другому рядку вхідних даних записано число N – кількість предметів. У наступних N рядках дано маси предметів, самі предмети перераховані в порядку спадання цінності (спочатку вказана маса найціннішого предмета, потім маса другого по цінності предмета і т.д.). Всі числа натуральні, сума ваги всіх предметів не перевищує $2*10^9$.

Технологія виконання

В даній задачі необхідно реалізувати принцип, який описаний в умові, а саме: беремо предмет, і якщо його маса не перевищує максимально дозволену масу, то кладемо його в рюкзак, інакше кладемо його до валізи.

1. Створіть новий файл за збережіть його під ім’ям «Завдання №3»

2. Нехай *vr* – маса рюкзака і *vv* – маса валізи. Беремо перший предмет з масою *r* і перевіряємо, якщо *vr*+*r* не перевищує **s**, то збільшуємо *vr* на *r*, інакше *vv* збільшуємо на *r*.

3. Розглянемо алгоритм розв’язування задачі:

- Ввести значення *S* та *N*, присвоїти значення 0 змінним *vr* і *vv*.
- За допомогою циклу та команди розгалуження обчислити *vr* і *vv*.
- Вивести результат.

4. Отримуємо програму:

```
s=int(input())
n=int(input())
vr=int(0)
vv=int(0)
for i in range(1, n+1):
    r=int(input())
    if (vr+r)<=s:
        vr=vr+r
    else:
        vv=vv+r
print(vr, vv)
```

5. Збережіть її та виконайте. Перевірте застосовуючи довільні набори даних.

6. Перегляньте результати та при потребі зробіть необхідні корективи.

Завдання №4

Створіть та збережіть програму під ім'ям «Завдання №4», після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано квадрат розміри сторін якого визначає користувач використовуючи цикл for.

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання №4»

2. Зважаючи на отриманий при виконанні попередніх завдань досвід, створення програми не має викликати труднощів.

3. Збережіть та виконайте програму для кількох значень a (довжини сторони квадрату)

4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте, виконавши програму.

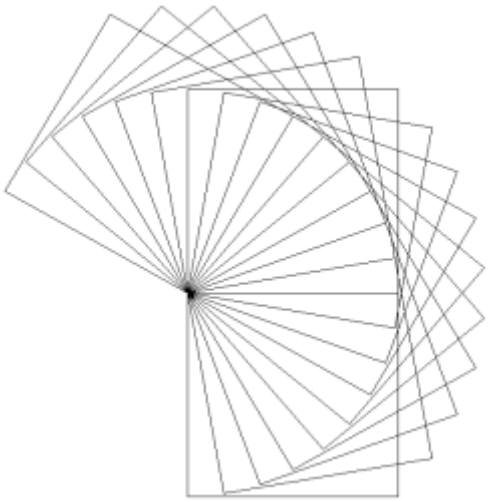
5. Ознайомтесь з одним з можливих варіантів розв'язання [24].

```
from turtle import*
reset()
a = int(input('Введіть довжину сторони квадрату: '))
for i in range(4):
    forward(a)
    right(90)
exitonclick()
```

Завдання для самостійного виконання

1. Створіть програму для обчислення суми кубів n перших натуральних чисел ($s = 1 + 23 + 33 + \cdots + n_3$).

2. Створіть програму для побудов орнаментів, що складаються з квадратів зі спільною вершиною повернутих на певний кут. Кількість квадратів, довжину сторони та кут повороту визначає користувач у процесі діалогу під час виконання програми.



3. Створіть програму для побудов штрихової лінії. Кількість штрихів та їх довжину визначає користувач у процесі діалогу під час виконання програми.

4. Знайти всі двозначні числа, в яких є цифра N або саме число ділиться на N.

5. Визначити кількість тризначних натуральних чисел, сума цифр яких дорівнює заданому числу N та вивести їх на екран.

6. Скласти програму обчислення суми кубів чисел від 25 до 55.

7. Серед двозначних чисел знайти ті, сума квадратів цифр яких ділиться на 13.

8. Написати програму пошуку двозначних чисел, таких, що якщо до суми цифр цього числа додати квадрат цієї суми, то вийде це число.

9. Квадрат тризначного числа закінчується трьома цифрами, які якраз і складають це число.

10. Написати програму пошуку чотиризначного числа, яке при діленні на 133 дає в залишку 125, а при діленні на 134 дає в залишку 111.

11. Знайти суму натуральних чисел з проміжку від A до B (B ≤ 100000), кратних 4 (значення змінних A і B вводяться з клавіатури).

12. Знайти суму натуральних чисел, більших 20, менших 100, кратних 3 і які закінчуються на 2, 4 або 8.

13. У тризначному натуральному числі закреслили першу цифру зліва, коли отримане двозначне число помножили на 7, то отримали дане число. Знайти це число.

14. Сума цифр тризначного натурального числа кратна 7, саме число також ділиться на 7. Знайти всі такі числа.

15. Серед чотиризначних натуральних чисел вибрати ті, у яких всі чотири цифри різні.

16. Серед двозначних натуральних чисел знайти ті, сума цифр яких дорівнює *n* (*0<n<18*) і число ділиться без залишку на числа *q*.

17. Дано чотиризначне натуральне число *n*. Викинути із запису числа *n* цифри 0 і 5, залишивши колишнім порядок інших цифр. Наприклад, з числа 1509 повинно вийти 19.

18. Натуральне число з *n* цифр є числом Армстронга, якщо сума його цифр, піднесених до *n*-го степеня, дорівнює самому числу (наприклад, 153 = 13 + 53 + 33). Отримати всі числа Армстронга, що складаються з трьох і чотирьох цифр.

19. Дано натуральне число *n* (*n≤1000000*). Знайти всі його дільники та їх суму.

20. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) семикутник (розмір та товщину сторони, колір заливки оберіть довільним чином). (Черепашка.)

21. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано правильний (рівносторонній) n-кутник довжину сторони та їх кількість якого визначає

користувач у процесі виконання програми (розмір та товщину сторони, колір заливки оберіть довільним чином) [24].

22. Відокремити аналітичним методом корінь рівняння $2x^3 - 17x + 8 = 0$. Реалізувати алгоритм у середовищі Python [2].

3.4.4. Цикл while

Цикл while має такий вигляд:

<Початкове значення>

while <Умова>:

<Інструкції>

<Приріст>

[else:

<Блок, що виконується, якщо не використовувався оператор break>]

Виконання інструкцій в циклі while продовжується до тих пір, доки умова виконується (логічний вираз після while істинний).

Послідовність роботи циклу while:

1. Змінній-лічильнику присвоюється початкове значення.

2. Перевіряється умова, якщо вона істинна, виконуються інструкції всередині циклу, інакше виконання циклу завершується.

3. Змінна-лічильник змінюється на величину, зазначену в параметрі <Приріст>.

4. Перехід до пункту 2.

5. Якщо всередині циклу не використовувався оператор break, то після завершення виконання циклу буде виконано блок в інструкції else. Даний блок не є обов'язковим.

У якості прикладу розглянемо програму за допомогою якої виводяться числа від 1 до 27, використовуючи цикл *while.*

```
i = 1            # <Початкове значення>
while i < 28:    # <Умова>
    print(i)     # <Інструкції>
    i=i+1        # <Приріст>
input()
```

Зауваження

Якщо <Приріст> не вказано, то цикл буде нескінченним. Щоб перервати нескінченний цикл, слід натиснути комбінацію клавіш <Ctrl> + <C>. В результаті генерується виключення KeyboardInterrupt, і виконання програми буде зупинено. Слід враховувати, що перервати таким чином можна тільки цикл, який виводить дані. За допомогою циклу while можна перебирати і елементи різних структур. Але в цьому випадку слід пам'ятати, що цикл while працює повільніше циклу for [24].

3.4.5. Оператор continue. Оператор break

Перехід на наступну ітерацію циклу. Оператор continue

Оператор *continue* дозволяє перейти до наступної ітерації (повторювання) циклу до завершення виконання всіх інструкцій всередині циклу. Як приклад виведемо всі числа від 1 до 50, крім чисел від 17 до 27 включно.

```
for i in range(1, 51):
    if 16 < i < 28:
        continue    # Переходимо до наступної ітерації
    print(i)
```

Переривання циклу. Оператор break

Оператор break дозволяє перервати виконання циклу достроково. Для прикладу виведемо всі числа від 1 до 27 ще одним способом.

```
i = 1
while True:
    if i > 27: break # Перериваємо цикл
    print(i)
    i = i + 1
input()
```

Тут ми в умові вказали значення True. У цьому разі висловлення всередині циклу будуть виконуватися нескінченно. Однак використання оператора break перериває його виконання, як тільки надруковано 27 рядків.

УВАГА!

Оператор *break* перериває виконання циклу, а не програми, іншими словами, далі буде виконана інструкція, наступна за циклом. Цикл *while* спільно

з оператором *break* зручно використовувати для отримання невизначеної заздалегідь кількості даних від користувача. Як приклад розглянемо програму для знаходження суми невизначеної кількості чисел [24].

```
print("Введіть слово 'стоп' для отримання результату")
summa = 0
while True:
    x = input("Введіть число: ")
    if x == "стоп": break        # Вихід з циклу
    x = float(x)                 # Перетворюємо рядок у число
    summa = summa+x
print("Сума чисел дорівнює:", summa)
input()
```

Процес введення 4 чисел і отримання суми виглядає так:

```
Введіть слово 'стоп' для отримання результату
Введіть число: 15
Введіть число: 25
Введіть число: 33
Введіть число: 17
Введіть число: стоп
Сума чисел дорівнює: 90.0
```

Контрольні запитання

1. Який синтаксис та правила виконання циклу while? Коли доцільно його використовувати?

2. Яке призначення оператора continue?

3. За допомогою якого оператора можна перервати виконання циклу while достроково?

Практичні завдання

Завдання №1

Складіть програму для обчислення найбільшого спільного дільника (НСД) двох натуральних чисел. Збережіть її під назвою «Завдання №1».

Технологія виконання

1. Створіть новий файл та збережіть його під ім’ям «Завдання №1».

2. Для створення програми використаємо алгоритм знаходження НСД, який полягає у наступному:

- Від більшого числа віднімаємо менше і різницю ставимо на місце більшого числа.

- Повторюємо цей процес доти, доки числа не стануть рівними.
- Це і буде НСД двох цих чисел.

3. Результат реалізації даного алгоритму на мові Python може бути, наприклад, таким:

```
a=int(input("a="))
b=int(input("b="))
m=a
n=b
while a!=b:
    if a>b:
        a=a-b
    else:
        b=b-a
print("НСД(", m,",", n,")=", a)
input()
```

4. Виконайте програму для кількох наборів даних.

5. Перегляньте результати виконання та при потребі зробіть необхідні корективи.

Завдання №2

Дано натуральне число *n*. Складіть програму для підрахунку кількості цифр даного числа. Збережіть програму з назвою «Завдання №2».

Технологія виконання

Кількість цифр у числі *n* невідомо, тому необхідно використовувати оператор *while*. Використання *for* вимагає або введення додаткових змін, або штучного виходу з циклу. Присвоїмо змінній *m* значення змінної *n*. Підрахунок кількості цифр почнемо з останніх цифр числа. Збільшимо лічильник цифр на одиницю *k*. Число *m* зменшимо в 10 разів, тим самим прибираючи з нього останню цифру (вже підраховану). Далі з отриманим числом виконаємо ще одну таку ж саму послідовність дій і т. д. доки число не стане рівним нулю. Розглянемо ручне «трасування». Нехай введене число 75490. Тоді кінцеве значення змінної *k* дорівнює 5. Кількість цифр – 5.

1. Створіть новий файл та збережіть його під ім’ям «Завдання №2».

2. Після проведеного аналізу створення відповідної програми не повинно викликати труднощів.

3. Один з можливих варіантів розв’язання:

```
n = int(input('Введіть натуральне число: '))
m = n
k = 0
while m!=0:
    m = m//10
    k = k+1
print('У числі', n, '-', k, 'цифр.')
```

4. Виконайте програму для кількох наборів даних.

5. Перегляньте результати виконання та при потребі зробіть необхідні корективи [24].

Завдання для самостійного виконання

1. Складіть програму для знаходження суми цифр натурального числа *n*.

2. Складіть програму для визначення того, скільки раз задана цифра зустрічається у натуральному числі *n*.

3. Складіть програму для знаходження найбільшої цифри заданого натурального числа *n*.

4. У багажник автомобіля вантажать овочі і фрукти з дачі: картоплю, капусту, моркву, яблука, груші та ін. Обсяг багажника дорівнює *V* л. Продукти кладуть послідовно, обсяг кожного вантажу відомий у літрах. Потрібно сказати в який момент (визначити номер вантажу) багажник переповниться.

5. Напишіть програму, яка буде підсумовувати числа, що вводяться з клавіатури до тих пір, доки вони від’ємні.

6. Створіть програму, яка буде підсумовувати числа, які вводяться з клавіатури до тих пір, доки вони парні.

7. Дано число натуральне число *n*. Надрукувати ті натуральні числа, квадрат яких не перевищує *n*.

8. Напишіть програму, яка запитує у користувача числа до тих пір, доки кожне наступне число більше попереднього. В кінці програма повідомляє, скільки чисел було введено.

9. Дано натуральне число, у якому всі цифри різні. Визначити порядковий номер його максимальної цифри, рахуючи номери від кінця числа.

10. Написати програму перевірки, чи є число *X* степенем числа 2 [24].

11. В перший день спортсмен пробіг *x* кілометрів, а потім він кожен день збільшував пробіг на 10% від попереднього значення. За даним числом у визначте номер дня, за який пробіг спортсмена складе не менше у кілометрів.

Формат вхідних даних:

Програма отримує на вхід дійсні числа *x* і *y*.

Формат вихідних даних:

Програма повинна вивести одне натуральне число.

12. Напишіть програму, яка переставляє цифри числа у зворотному порядку.

Формат вхідних даних:

Задано єдине число *N*.

Формат вихідних даних:

Необхідно вивести цифри даного числа у зворотному порядку.

13. Виконавець «Поділяєць» перетворює натуральні числа. У нього є дві команди: «Відняти 1» і «Розділити на 2», перша команда зменшує число на 1, друга команда зменшує число в два рази, якщо воно парне, інакше виконується помилка. Дано два натуральних числа A і B (A > B). Напишіть алгоритм для Поділяйця, який перетворює число A в число B і при цьому містить мінімальну кількість команд. Команди алгоритму потрібно виводити по одному в рядку, перша команда позначається, як -1, друга команда як: 2.

Формат вхідних даних:

Вводяться два натуральних числа A і B.

Формат вихідних даних:

Виведіть відповідь до задачі [35].

3.5. Функції у Python

Зазвичай реалізація складних задач містить великі фрагменти коду. Зручним способом організувати об'ємний фрагмент коду в більш зручні фрагменти є створення функцій. Функції в Python – це основа при написанні програм.

Іноді функцію порівнюють з так званим «чорним ящиком», тобто коли відомо, що на вході і що при цьому на виході, а середина «чорного ящика» часто буває прихованою.

Існує велика кількість вбудованих функцій. Наприклад функція *abs()*, приймає на вхід один аргумент, а на виході повертає абсолютне значення об'єкта.

```
>>> abs(–9)
9
```

Результат виклику функції можна присвоїти змінній та використовувати його як операнд математичних виразів, тобто утворювати більш складні вирази.

Крім написання складних математичних виразів, Python дозволяє передавати результати виконання функцій і іншим функціям (як аргументи) без використання додаткових змінних.

```
>>> print(abs(–15))
15
```

Спочатку визначається абсолютне значення цілого числа -15, а потім за допомогою функції print() виводиться на екран результат розрахунку. Можна також реалізовувати і власні функції.

Функція – це визначений та іменований фрагмент коду, відокремлений від інших структур. Функції можуть приймати будь-яку кількість вхідних параметрів і повертати будь-яку кількість результатів відповідно.

З функцією можна зробити такі дві речі:

1. Визначити.

2. Викликати.

Щоб визначити функцію, застосовують таку конструкцію:

```
def Ім_я_функції (аргументи):
    Функція
```

Назви (ім’я) функцій підкоряються тим же правилам, що й імена змінних (вони повинні починатися з літери або символ «_» і містити тільки букви, цифри або «_»).

Функція може не містити параметри, проте круглі дужки все одно необхідно вказувати:

```
def Nothing():
  pass
```

Тіло функції відділяється пробілами. Використання оператору *pass* вказує на те, що функція нічого не робить.

Щоб викликати функцію вказується її ім’я та дужки з відповідними параметрами (аргументами) або без них: *Nothing()*.

Всі дії в коді програми виконуються послідовно зверху до низу. Це означає, що перш ніж використовувати ідентифікатор, його слід попередньо оголосити, присвоївши йому певне значення. Тому визначення функції має бути розташоване безпосередньо перед викликом функції.

Визначимо та викличемо функцію, яка не має параметрів і виводить на екран одне слово:

```
def Hello():
  print("Salute!")
Hello()
```

Одержимо:

```
Salute!
```

Коли викликається функція, вданому випадку – *Hello()*, Python виконує код, розташований всередині її опису (тіла). Тут результат функції – це виведення одного слова, після чого вона повертає управління основній програмі.

Функція може приймати і параметри та повертати значення. Параметри функції – це звичайні змінні, які функція використовує для внутрішніх розрахунків. Якщо параметрів декілька, то вони перераховуються через кому.

Формальні параметри – це параметри, які вказуються при оголошенні функції.

Фактичні параметри – це параметри, які безпосередньо передаються в функцію при її виклику.

```
def print_Num(n):
  for var in range(n):
    print(var)
N = int(input("Введіть кількість елементів N : "))
print_Num(N)
```

Результатом запуску коду буде:

```
Введіть кількість елементів N : 2
0
1
```

Значення, які передаються в функцію при її виклику, називаються *аргументами*. Коли функція викликається разом зі аргументами, їх значення копіюються у відповідні параметри в тілі функції.

Існують функції, які просто щось виконують, наприклад функція *print()* виводить на екран певні значення:

```
>>> n = 15
>>> print(n)
```

Отримаємо:

```
15
```

А є функції, які повертають певне значення, що може бути присвоює змінній, наприклад функція *input()*:

```
>>> a = input("Введіть слово: ")
Введіть слово: Привіт!
>>> print(a)
```

Одержимо:

```
Привіт!
```

При потребі повернути результат роботи функції в програму, з якої вона викликалася, з метою її подальшого оброблення використовується команда *return*.

Вираз, що стоїть після *return*, буде повертатися як результат виклику функції.

Без аргументів *return* використовується для виходу з функції, бо інакше вихід відбудеться при досягненні кінця тіла функції.

В Python функції можуть повертати кілька значень одночасно.

```
def Roots(var_a, var_b, var_c):
  var_D = var_b**2 - 4*var_a*var_c
  import math
  x_1 = (-var_b + math.sqrt(var_D))/(2*var_a)
  x_2 = (-var_b - math.sqrt(var_D))/(2*var_a)
  return x_1, x_2
print(Roots(1.0, 0, -1))
```

Результатом запуску коду буде:

```
(1.0, -1.0)
```

Окрім того, результати виконання функції можна присвоювати відразу кільком змінним:

```
x_1, x_2 = Roots(1.0, 0, -1)
print("x_1 =", x_1, "\nx_2 =", x_2)
```

Результатом запуску буде:

```
x_1 = 1.0
x_2 = -1.0
```

Всередині тіла функції може міститися довільна кількість операторів *return*. Однак він спрацює лише один раз. Наприклад:

```
var_C = "Зелений"
def Traffic_lights(var_C):
  if var_C == "Червоний":
    return print("СТОП!")
```

```
  elif var_C == "Зелений":
    return print("- - >")
  elif var_C == "Жовтий":
    return print("Приготуватись!")
  else:
    return print("Світлофор вийшов з ладу!")
Traffic_lights(var_C)
```

Результат:

```
- - >
```

Функція може набувати будь-яку кількість значень, тобто аргументів (включаючи нуль), причому будь-якого типу. Вона може повертати будь-яку кількість результатів (також нуль) теж будь-якого типу. Якщо функція не викликає *return* явно, буде отримано результат типу – *None* [6].

Контрольні запитання

1. Навіщо використовуються функції користувача?
2. Як описується функція та яким має бути її ім’я?
3. Коли та навіщо використовується інструкція return?
4. Які правила розташування визначень функцій користувача?

Практичні завдання

Завдання №1

Складіть програму для знаходження найбільшого спільного дільника п’ятьох натуральних чисел. Збережіть програму під назвою «Завдання №1».

Технологія виконання

1. Створіть новий файл та збережіть його під ім’ям «Завдання №1».
2. Створимо функцію для знаходження НСД двох натуральних чисел, перед кодом програми додамо функцію, яку викликатимемо в потрібний нам момент. Приклад коду може бути таким:

```
def ncd(a, b):
    while a!=b:
        if a>b:
            a=a-b
        else:
            b=b-a
    return a
```

3. Наступна розробка програми не являє собою особливих складнощів. Допрацюйте код програми, збережіть її та виконайте для різних наборів даних.

4. Перегляньте результати виконання та при потребі зробіть необхідні корективи. Перевірте виконавши програму.

5. У випадку виникнення труднощів, перевірте код.

```
def ncd(a, b):
    while a!=b:
        if a>b:
            a=a-b
        else:
            b=b-a
    return a
n1=int(input("n1="))
n2=int(input("n2="))
n3=int(input("n3="))
n4=int(input("n4="))
n5=int(input("n5="))
ncd5=ncd(n1, n2)
ncd5=ncd(n3, ncd5)
ncd5=ncd(n4, ncd5)
ncd5=ncd(n5, ncd5)
print("НСД 5-ти чисел дорівнює:", ncd5)
input()
```

Завдання №2

Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано *n* квадратів, довжиною

сторони *b* пікселів та на відстані с один від одного (значення *n, b* та *c* визначає користувач у процесі діалогу користувача з комп'ютером під час виконання програми). Після виконання побудов виконавець Черепашка повертається у початкове положення. Збережіть програму під ім'ям «Завдання №2».

Технологія виконання

1. Відкрийте інтегроване середовище розробки IDLE.

2. Створіть новий файл та збережіть його під ім’ям «Завдання №2».

3. Для побудови квадрату з певною довжиною сторони використаємо функцію з параметром:

```
def sq(a):
    for i in range(4):
        down()
        forward(a)
        right(90)
        up()
```

4. Щоб побудувати потрібну кількість квадратів застосуємо цикл for:

```
for i in range(n):
    sq(b)
    fd(b+c)
```

5. Запустіть створену програму на виконання.

6. Перегляньте результати та при потребі зробіть необхідні корективи [24].

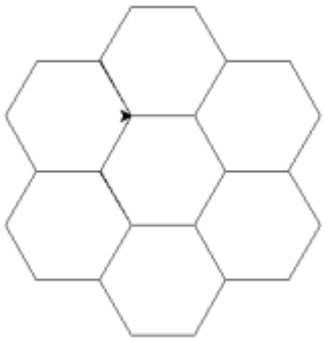
Завдання для самостійного виконання

1. Складіть програму для знаходження НСК 5-ти натуральних чисел.

2. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано *n* правильних трикутників, довжиною сторони *b* пікселів та на відстані **c** один від одного (значення *n*, *b* та *c* визначає користувач у процесі діалогу користувача з комп’ютером під час виконання програми).

3. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано соти (складаються з правильних шестикутників), розмір шестикутників оберіть на ваш розсуд.

4. Створіть програму, після виконання якої, на полотні Python Turtle Graphics за допомогою модуля Черепашка буде побудовано соти (складаються з правильних шестикутників). Розміри шестикутників визначає користувач у процесі діалогу користувача з комп’ютером під час виконання програми [24].



5. Напишіть функцію, що отримує як вхідну інформацію довжини двох катетів прямокутного трикутника і повертає довжину гіпотенузи, обчислену за теоремою Піфагора. У головній програмі має відбуватись запит довжин сторін у користувача, виклик функції та вивід на екран отриманого результату [39].

3.6. Списки та кортежі у Python

Списки та кортежі – це набори об’єктів. Кожен елемент набору містить лише посилання на об’єкт. З цієї причини списки і кортежі можуть містити об’єкти довільного типу даних і мати необмежену ступінь вкладеності. Позиція елемента в наборі задається індексом. Зверніть увагу на те, що нумерація елементів починається з 0, а не з 1.

У мові Python списки є аналогом масивів у інших мовах програмування, але вони мають більш широкі можливості. З одного боку, списки не обмежені одним типом елементів, з іншого, розмір списків не обмежений, завдяки чому кількість елементів списку можна збільшувати та зменшувати по мірі необхідності [24].

Списки та кортежі є впорядкованими послідовностями елементів. Як і всі послідовності, вони підтримують звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

Списки відносяться до змінюваних типів даних. Це означає, що ми можемо не тільки отримати елемент за індексом, а й змінити його. Кортежі відносяться

до незмінних типів даних. Іншими словами, можна отримати елемент за індексом, але змінити його не можна.

Створення списку

Створити список можна наступними способами:

• за допомогою функції list([<Послідовність>]). Функція дозволяє перетворити будь-яку послідовність в список. Якщо параметр не вказано, то створюється порожній список.

Приклад:

```
>>> list() # Створюємо порожній список.
[]
>>> list('Рядок') # Перетворюємо рядок у список.
['Р', 'я', 'д', 'о', 'к']
>>> list((7, 2, 5, 5, 9)) # Перетворюємо кортеж у список.
[7, 2, 5, 5, 9]
```

• вказавши всі елементи списку всередині квадратних дужок:

```
>>> arr=[5, 1, 'py', 8, 't']
>>> arr
[5, 1, 'py', 8, 't']
```

• заповнивши список поелементно за допомогою методу append():

```
>>> arr=[] # Створюємо порожній список.
>>> arr.append(7) # Додаємо елемент 7(індекс 0).
>>> arr.append(-10) # Додаємо елемент -10(індекс 1).
>>> arr
[7, -10]
```

Наведемо фрагменти коду з прикладами введення значень елементів списку з клавіатури.

Фрагмент програми, при виконанні якого буде введено значення 4-х елементів створеного списку.

```
A=[]
for i in range(4):
    print("A[",i,"]=",end="")
    A.append(input())
print(A)
```

Фрагмент програми, при виконанні якого буде введено n (визначається користувачем) елементів створеного списку A.

```
A=[]
n=int(input("n="))
for i in range(n):
    print("A[",i,"]=",end="")
    A.append(input())
print(A)
```

Ввести послідовність цілих чисел, які записані у стрічку через пробіл можна ще за допомогою таких інструкцій:

- `A = list(map(int, input().split()))`, в інструкції використовується функція map та метод split для роботи з рядками;
- `A = [int(i) for i in input().split()]`, тут застосовується list comprehension, генерування списку [24].

3.6.1. Основні операції над списками та кортежами

Звернення до елементів списку здійснюється за допомогою квадратних дужок, в яких вказується індекс елемента. Нумерація елементів списку починається з нуля.

Приклад:

```
>>> arr=[7, 8, 'm']
>>> arr[0]
7
>>> arr[2]
'm'
```

Індекси можуть бути від'ємними, у цьому випадку нумерація відбувається з кінця (кількість елементів списку + від'ємний індекс).

Нехай:

```
>>> a=[5,7,3,0,1]
```

Маємо:

```
>>> a[-1]
1
>>> a[-4]
7
```

Розглянувши всі індекси для списку **a**, отримаємо:

список ***a***	5	7	3	0	1

індекси	*a*[0]	*a*[1]	*a*[2]	*a*[3]	*a*[4]
індекси	*a*[-5]	*a*[-4]	*a*[-3]	*a*[-2]	*a*[-1]

За допомогою позиційного присвоювання можна присвоїти значення елементів списку будь-яким змінним. Кількість елементів праворуч і ліворуч від оператора = має збігатися, інакше буде виведено повідомлення про помилку:

```
>>> x, y = [7, 2, 10]
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    x, y = [7, 2, 10]
ValueError: too many values to unpack (expected 2)
```

В Python 3 при позиційному присвоюванні перед однією із змінних зліва від оператора = ви можете вказати зірочку. У цієї змінної буде зберігатися список, що складається з «зайвих» елементів. Якщо таких елементів немає, то список буде порожнім:

```
>>> x, y, *z = [7, 21, 3, 14, 5]
>>> x
7
>>> z
[3, 14, 5]
```

Оскільки нумерація елементів списку починається з 0, індекс останнього елемента буде на одиницю менше кількості елементів. Отримати кількість елементів списку дозволяє функція *len():*

```
>>> arr = [7, 2, 8, 7, 5]
>>> len(arr)
5
```

Якщо елемент, що відповідає вказаному індексу, відсутній у списку, то збуджується виключення *IndexError:*

```
>>> arr = [7, 12, 8, 7, 15]
>>> arr[5]
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    arr[5]
IndexError: list index out of range
```

Оскільки списки відносяться до змінюваних типів даних, то ми можемо змінити елемент за індексом:

```
>>> arr = [7, 12, 8, 17, 1]
>>> arr[0] = 100
>>> arr
[100, 12, 8, 17, 1]
```

Крім того, списки підтримують операцію вилучення зрізу, яка повертає зазначений фрагмент списку. Формат операції:

[<Початок>: <Кінець>: <Крок>]

Всі параметри є необов'язковими. Якщо параметр *<Початок>* не вказано, то використовується значення 0. Якщо параметр *<Кінець>* не вказано, то повертається фрагмент до кінця списку. Слід також зауважити, що елемент з індексом, зазначеним у цьому параметрі, не входить до фрагменту, що повертається. Якщо параметр *<Крок>* не вказано, то використовується значення 1. В якості значення параметрів можна вказати від'ємні значення [24].

Тепер розглянемо кілька прикладів. Спочатку отримаємо поверхневу копію списку:

```
>>> arr = [11, 72, 35, 4, 5]
>>> k=arr[:]    #Створюємо поверхневу копію.
>>> k
[11, 72, 35, 4, 5]
```

Створимо копію списку у якій елементи розташовані у зворотному порядку:

```
>>> m=arr[::-1]
>>> m
[5, 4, 35, 72, 11]
>>> arr
[11, 72, 35, 4, 5]
```

Виведемо список без першого і останнього елементів:

```
>>> arr[:-1] #Без останнього елемента.
[11, 72, 35, 4]
>>> arr[1:] #Без першого елемента.
[72, 35, 4, 5]
```

Отримаємо перші два елементи списку:

```
>>> arr[0: 2] #Символ з індексом 2 не входить до діапазону.
[11, 72]
```

А тепер отримаємо останній елемент:

```
>>> arr[-1:] # Останній елемент списку.
[5]
```

І, нарешті, виведемо фрагмент від другого елементу до четвертого включно:

```
>>> arr[1:4]
[72, 35, 4]
```

З’єднати два списки у один дозволяє оператор +. Результатом об’єднання буде новий список:

```
>>> arr1=[7,6,15]
>>> arr2=[17,8,15,25]
>>> arr3=arr2+arr1
>>> arr3
[17, 8, 15, 25, 7, 6, 15]
```

Замість оператора + можна використовувати оператор +=. Слід враховувати, що у цьому випадку елементи додаються до поточного списку:

```
>>> arr = [1, 2, 3, 4, 5]
>>> arr += [6, 7, 8, 9]
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Крім розглянутих операцій, списки підтримують операцію повторення і перевірку на входження. Повторити список вказану кількість разів можна за допомогою оператора *, а виконати перевірку на входження елемента у список дозволяє оператор *in:*

```
>>> [7, 5]*3
[7, 5, 7, 5, 7, 5]
>>> 7 in [8, 3, 7, 5]
True
```

3.6.2. Багатовимірні списки

Будь-який елемент списку може містити об’єкт довільного типу. Наприклад, елемент списку може бути числом, рядком, списком, кортежем, тощо. Створити вкладений список можна, наприклад, так:

```
>>> arr = [ [7, 2, 4], [14, 5, 9], [7, 10, 19] ]
```

Як ви вже знаєте, вираз всередині дужок може розташовуватися на декількох рядках. Отже, попередній приклад можна записати інакше:

```
>>> arr = [
        [7, 2, 4],
        [14, 5, 9],
        [7, 10, 19]
        ]
```

Щоб отримати значення елемента у вкладеному списку, слід вказати два індекси:

```
>>> arr[2][1]
10
```

Елементи вкладеного списку також можуть мати елементи довільного типу. Кількість вкладень необмежена. Таким чином, ми може створити об’єкт будь-якого ступеня складності. В цьому випадку для доступу до елементів вказується кілька індексів поспіль [24].

Приклад:

```
>>> arr = [ [7, ["a", "b"], 3], [4, 5, 6], [17, 8, 9] ]
>>> arr[0][1][1]
'b'
```

Перебір елементів списку

Перебрати всі елементи списку можна за допомогою циклу for:

```
>>> arr = [11, 25, 3, 4, 15]
>>> for i in arr: print(i, end=" ")

11 25 3 4 15
```

Слід зауважити, що змінну і всередині циклу можна змінити, але якщо вона посилається на незмінний тип даних (наприклад, число або рядок), то це не відіб’ється на вихідному списку.

Щоб отримати доступ до кожного елементу, можна, наприклад, скористатися функцією *range()* для генерації індексів.

```
arr = [17, 2, 37, 24]
for i in range(len(arr)):
    arr[i] *= 2
print(arr)
```

Результат виконання:

```
[34, 4, 74, 48]
```

Крім того, перебрати елементи можна за допомогою циклу *while*. Але в цьому випадку слід пам’ятати, що цикл *while* працює повільніше циклу *for*. Як приклад помножимо кожний елемент списку на 3, використовуючи цикл *while:*

```
arr = [11, 2, 5, 1]
i, c = 0, len(arr)
while i < c:
    arr[i] *= 3
    i += 1
print(arr)
```

Результат виконання:

```
[33, 6, 15, 3]
```

Додавання і видалення елементів списку

Для додавання і видалення елементів списку використовуються наступні методи:

• append(<Об'єкт>) – додає один об'єкт у кінець списку. Метод змінює поточний список і нічого не повертає. Приклад:

```
>>> arr = [11, 2, 3, 7]
>>> arr.append(5) # Додаємо число
>>> arr
[11, 2, 3, 7, 5]
>>> arr.append([5, 6]) # Додаємо список
>>> arr
[11, 2, 3, 7, 5, [5, 6]]
```

• extend(<Послідовність>) – додає елементи послідовності у кінець списку. Метод змінює поточний список і нічого не повертає. Приклад:

```
>>> arr = [1, 2, 3]
>>> arr.extend([4, 5, 6])
>>> arr.extend((7, 8, 9))
>>> arr.extend("Python")
>>> arr
[1, 2, 3, 4, 5, 6, 7, 8, 9, 'P', 'y', 't', 'h', 'o', 'n']
```

• Додати кілька елементів можна за допомогою операції конкатенації (+) або оператора +=:

```
>>> arr=[6, 12, 3]
>>> arr+[14, 5, 9] #Повертає новий список
[6, 12, 3, 14, 5, 9]
>>> arr+=[4, 5, 6] #Змінює поточний список
>>> arr
[6, 12, 3, 4, 5, 6]
```

• *insert (<Індекс>, <Об'єкт>)* – додає один об'єкт в зазначену позицію. Інші елементи зміщуються. Метод змінює поточний список і нічого не повертає. Приклади:

```
>>> arr = [11, 5, 13]
>>> arr.insert(0, 0) #Додаємо 0 на початок списку
>>> arr
[0, 11, 5, 13]
>>> arr.insert(-2, 22) #Можна вказати від'ємні числа
>>> arr
[0, 11, 22, 5, 13]
>>> arr.insert(3, 77) #Вставляємо 77 в позицію 3
>>> arr
[0, 11, 22, 77, 5, 13]
>>> arr.insert(6, [44, 55]) #Додаємо список
>>> arr
[0, 11, 22, 77, 5, 13, [44, 55]]
```

• pop ([<Індекс>]) – видаляє елемент, розташований за вказаним індексом, і повертає його. Якщо індекс не вказано, то видаляє і повертає останній елемент списку. Якщо елемента з вказаним індексом немає або список порожній, збуджується виключення IndexError.

Приклади:

```
>>> arr=[7, 4, 22, 77, 14]
>>> arr.pop()
14
>>> arr
[7, 4, 22, 77]
>>> arr.pop(0)
7
>>> arr
[4, 22, 77]
>>> arr.pop(1)
22
>>> arr
[4, 77]
```

• Видалити елемент списку дозволяє також оператор *del:*

```
>>> arr=[5, 6, 7, 8, 9]
>>> del arr[4]
>>> arr
[5, 6, 7, 8]
>>> del arr[1:2]
>>> arr
[5, 7, 8]
>>> del arr[1:3]
>>> arr
[5]
```

• *remove(<Значення>)* – видаляє перший елемент, що містить вказане значення. Якщо елемент не знайдений, то збуджується виключення ValueError. Метод змінює поточний список і нічого не повертає. Приклади:

```
>>> arr=[7, 5, 99, 100, 12, 99]
>>> arr.remove(99)
>>> arr
[7, 5, 100, 12, 99]
```

Якщо необхідно видалити всі повторювані елементи списку, то можна список перетворити у множину, а потім множину назад перетворити в список. Зверніть увагу на те, що список має містити тільки незмінні об’єкти (наприклад, числа, рядки або кортежі). В іншому випадку порушується виключення TypeError [24].

Приклад:

```
>>> arr = [1, 2, 3, 1, 1, 2, 2, 3, 3]
>>> s = set(arr)
>>> s
{1, 2, 3}
>>> arr = list(s)
>>> arr
[1, 2, 3]
```

Пошук елемента у списку

Виконати перевірку на входження елемента в список дозволяє оператор *in*. Якщо елемент входить до списку, то повертається значення *True*, в іншому випадку – *False*.

Приклад:

```
>>> arr = [7, 5, 100, 12, 99]
>>> 100 in arr
True
>>> 77 in arr
False
```

Проте, оператор in не дає ніякої інформації про місцезнаходження елемента всередині списку. Щоб дізнатися індекс елементу всередині списку, слід скористатися методом *index()*. Формат методу:

index(<Значення> [, <Початок> [, <Кінець>]])

Метод *index()* повертає індекс елемента, що має вказане значення. Якщо значення не входить до списку, то збуджується виключення *ValueError*. Якщо другий і третій параметри не зазначені, то пошук буде проводитися з початку списку.

Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7]
>>> arr.index(3)
0
>>> arr.index(9, 1, 5)
2
>>> arr.index(6)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    arr.index(6)
ValueError: 6 is not in list
```

Дізнатися загальну кількість елементів із зазначеним значенням дозволяє метод *count(<Значення>).* Якщо елемент не входить в список, то повертається значення 0. Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7, 3]
>>> arr.count(3)
3
>>> arr.count(7)
2
>>> arr.count(6)
0
```

За допомогою функцій max() і min() можна дізнатися максимальне і мінімальне значення списку відповідно. Приклад:

```
>>> arr=[3, 4, 9, 3, 5, 7, 7, 3]
>>> max(arr)
9
>>> min(arr)
3
```

Перевертання та перемішування списку

Метод *reverse()* змінює порядок проходження елементів списку на протилежний. Метод змінює поточний список і нічого не повертає.

Приклад:

```
>>> arr = [3, 4, 9, 3, 5, 7, 7, 3]
>>> arr.reverse()
>>> arr
[3, 7, 7, 5, 3, 9, 4, 3]
```

Функція *shuffle(<Список> [, <Число від 0.0 до 1.0>])* з модуля *random* «перемішує» список випадковим чином. Функція перемішує сам список і нічого не повертає. Якщо другий параметр не вказано, то використовується значення, що повертається функцією *random().*

Приклад:

```
>>> import random
>>> arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> random.shuffle(arr)
>>> arr
[7, 4, 9, 6, 3, 10, 2, 5, 8, 1]
```

Вибір елементів випадковим чином

Отримати елементи зі списку випадковим чином дозволяють наступні функції з модуля random:

• *choice(<Послідовність>)* – повертає випадковий елемент з будь-якій послідовності (рядку, списку, кортежу):

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> import random
>>> random.choice(arr)
7
>>> random.choice(arr)
4
```

• *sample(<Послідовність>, <Кількість елементів>)* – повертає список із зазначеною кількістю елементів. У цей список потраплять елементи з послідовності, вибрані випадковим чином. У якості послідовності можна вказати будь-які об'єкти, що підтримують ітерації. Приклад:

```
>>> arr = [1, 2, 3, 4, 5, 6, 7]
>>> import random
>>> random.sample(arr, 4)
[7, 2, 1, 3]
```

Сортування списку

Відсортувати список дозволяє метод *sort()*. Метод має такий вигляд: *sort([key = None] [, reverse = False])*

Всі параметри є необов'язковими. Метод змінює поточний список і нічого не повертає. Відсортуємо список по зростанню з параметрами за замовчуванням:

```
>>> arr = [4, 1, 9, 14, 5, 7, 5]
>>> arr.sort()
>>> arr
[1, 4, 5, 5, 7, 9, 14]
```

Щоб відсортувати список за спаданням, слід в параметрі reverse вказати значення *True:*

```
>>> arr = [4, 1, 9, 14, 5, 7, 5]
>>> arr.sort(reverse=True)
>>> arr
[14, 9, 7, 5, 5, 4, 1]
```

При сортуванні фактично порівнюються два об'єкти, і який з них «менше», ставиться перед тим, який «більше». Поняття «більше» і «менше» досить умовні.

І якщо для чисел все просто - числа розставляються в порядку зростання, то для рядків і інших об’єктів ситуація складніша. Зокрема, рядки оцінюються по перших символах. Якщо перші символи рівні, оцінюються другі символи і так далі. При чому цифровий символ вважається «менше», ніж алфавітний заглавний символ, а заглавний символ вважається меншим, ніж рядковий. Детальніше ми це розглянемо при вивченні рядків [24].

Таким чином, якщо в списку поєднуються рядки з верхнім та нижнім регістром, то ми можемо отримати не зовсім коректні результати, так як для нас рядок "bob" повинен стояти до рядка "Tom". І щоб змінити стандартну поведінку сортування, ми можемо передати в метод *sort()* в якості параметра функцію для зміни регістру символів у параметрі key:

```
>>> users = ["Tom", "bob", "alice", "Sam", "Bill"]
>>> users.sort(key=str.lower)
>>> users
['alice', 'Bill', 'bob', 'Sam', 'Tom']
```

Заповнення списку числами

Створити список, що містить діапазон чисел, можна за допомогою функції *range()*. Функція повертає об’єкт, що підтримує ітерації. Щоб з цього об’єкта отримати список, слід скористатися функцією *list()*.

Як приклад заповнимо список числами від 0 до 9:

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Створимо список, що складається з діапазону чисел від 1 до 14:

```
>>> list(range(1, 15))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

А тепер список парних чисел з діапазону від 2 до 14:

```
>>> list(range(2, 15, 2))
[2, 4, 6, 8, 10, 12, 14]
```

Тепер змінимо порядок чисел на протилежний:

```
>>> list(range(14, 0, -1))
[14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Якщо необхідно отримати список з випадковими числами (або випадковими елементами з іншого списку), то можна скористатися функцією *sample(<Послідовність>, <Кількість елементів>)* з модуля random.

Приклад:

```
>>> import random
>>> arr = [11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25]
>>> random.sample(arr, 5)
[91, 77, 11, 10, 25]
>>> random.sample(range(700), 3)
[305, 347, 645]
```

Створити список можна також за допомогою list comprehension (генерування (компонування) списку).

Нехай, наприклад, потрібно створити список із 20 випадкових цілих чисел таких що $-7 \le A[i] \le 7$. Тоді відповідна команда та результат її виконання можуть мати наступний вигляд:

```
>>> from random import randint
>>> A = [randint(-7, 7) for i in range(20)]
>>> A
[6, 4, -3, -1, -1, -3, -2, -2, 1, 5, 7, 7, 2, 2, -7, -5, 1, -6, 0, -2]
```

Якщо ж, наприклад, необхідно створити список із 5 випадкових дійсних чисел таких що $-7 \le A[i] \le 7$, то у цьому випадку відповідна команда та результат її виконання можуть мати наступний вигляд:

```
>>> from random import uniform
>>> A=[uniform(-7, 7) for i in range(5)]
>>> A
[6.047908377920264, 5.379689783298021, 1.02030022022792,
-5.9282700633688235, 5.49613669358814]
```

Кортежі

Кортежі, так само як і списки, є впорядкованими послідовностями елементів. Кортежі багато в чому аналогічні спискам, але мають одну дуже важливу відмінність – змінити кортеж не можна. Можна сказати, що кортеж – це список, доступний «тільки для читання» [24].

Наведемо причини, коли варто використовувати кортежі замість списків. Першою причиною є можливість захисту даних від випадкової зміни (захист від дурня). Якщо ми отримали набір даних, і є необхідність опрацьовувати його без зміни даних, то це як раз той випадок, коли доцільно використати кортеж.

Другою причиною є те, що кортежі в пам’яті займають менший об’єм у порівнянні зі списками.

Створити кортеж можна за допомогою таких дій:

• вказавши всі елементи через кому всередині круглих дужок (або без дужок):

```
>>> k1 = () #Створюємо порожній кортеж
>>> k2 = (5,) #Створюємо кортеж з одного елементу
>>> k3 = ('a', 'v', 44, 'Пітон') #Створюємо кортеж з 4 елементів
>>> k1, k2, k3
((), (5,), ('a', 'v', 44, 'Пітон'))
```

Зверніть особливу увагу на другий рядок прикладу. Щоб створити кортеж з одного елемента, необхідно в кінці вказати кому. Саме коми формують кортеж, а не круглі дужки. Якщо всередині круглих дужок немає ком, то буде створено об’єкт іншого типу. Пам’ятайте, що будь-який вираз в мові Python можна взяти у круглі дужки, а щоб отримати кортеж, необхідно поставити коми [24].

• за допомогою функції *tuple([<Послідовність>])*. Функція дозволяє перетворити будь-яку послідовність в кортеж. Якщо параметр не вказано, то створюється порожній кортеж.

Приклад:

```
>>> tuple("String")
('S', 't', 'r', 'i', 'n', 'g')
>>> arr = [11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25]
>>> tuple(arr)
(11, 2, 13, 4, 55, 6, 77, 8, 91, 10, 25)
```

Позиція елемента в кортежі задається індексом. Зверніть увагу на те, що нумерація елементів кортежу (як і списку) починається з 0, а не з 1. Як і всі послідовності, кортежі підтримують звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

Приклад:

```
>>> k = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
>>> k[0]
1
>>> k[::-1]
(10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
>>> k[3:5]
(4, 5)
>>> 8 in k
True
>>> 15 in k
False
>>> (11, 27, 7) * 3
(11, 27, 7, 11, 27, 7, 11, 27, 7)
>>> (6, 0, 4)+(9, 0, 5, 7)
(6, 0, 4, 9, 0, 5, 7)
```

Кортежи відносяться до незмінних типів даних. Іншими словами, можна отримати елемент за індексом, але змінити його не можна:

```
>>> k =(7, 3, 9)
>>> k[0]
7
>>> k[0] = 55
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    k[0] = 55
TypeError: 'tuple' object does not support item assignment
```

Отримати кількість елементів кортежу дозволяє функція *len():*

```
>>> k =(7, 3, 9, 15, 7)
>>> len(k)
5
```

Кортежі підтримують два методи:

• *index(<Значення>[, <Початок>[, <Кінець>]])* – повертає індекс елемента, що має вказане значення. Якщо значення не входить до кортежу, збуджується виключення ValueError. Якщо другий і третій параметри не зазначені, то пошук буде проводитися в усьому кортежі. Приклад:

```
>>> k = (7, 21, 5, 2, 9)
>>> k.index(21)
1
>>> k.index(2, 1, 3)
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    k.index(2, 1, 3)
ValueError: tuple.index(x): x not in tuple
>>> k.index(2, 1, 4)
3
```

• *count(<Значення>)* – повертає кількість елементів із зазначеним значенням. Якщо елемент не входить в кортеж, то повертається значення 0. Приклад:

```
>>> k = (2, 2, 7, 2, 1, 5)
>>> k.count(2)
3
>>> k.count(7)
1
>>> k.count(77)
0
```

Інших методів у кортежів немає. Щоб здійснювати операції над кортежами, слід скористатися вбудованими функціями, призначеними для роботи з послідовностями, які ми розглядали при вивченні списків [24].

Контрольні запитання

1. Що спільне та чим відрізняються кортежі та списки у мові Python?
2. Перерахуйте способи створення списків у Python?
3. Яким чином здійснюється звернення до елементів списку в мові програмування Python?
4. З якого числа починається нумерація елементів списку в мові Python?
5. Яка функція в мові Python дозволяє отримати кількість елементів списку?
6. Як можна змінити значення елемента списку?
7. Яким чином у мові Python реалізовані багатовимірні списки?
8. Які методи використовуються для додавання та вилучення елементів списку?
9. Що можна зробити за допомогою методу index()?
10. Який оператор дозволяє виконати перевірку на входження елемента до списку?
11. Яке призначення методу reverse()?
12. Що можна зробити за допомогою функції shuffle() з модуля random?

13. Назвіть функції з модуля random за допомогою яких можна отримати елементи зі списку випадковим чином.

14. Який метод дозволяє відсортувати список?

15. Які функції та яким чином можна використовувати для заповнення списку числами?

16. Яка найважливіша відмінність кортежів від списків?

17. Які є способи створення кортежів?

18. З якого числа починається нумерація елементів кортежу в мові Python?

19. Які методи підтримують кортежі починаючи з Python 2.6? Яке їх призначення?

Практичні завдання

Завдання №1

Знайти суму додатних елементів одновимірного масиву (таблиці) всі елементи якого є цілими числами. Збережіть програму під ім'ям «Завдання №1»

Технологія виконання

1. Створіть новий файл за збережіть його під ім'ям «Завдання №1».

2. Для розв'язування цієї задачі використаємо тип даних список.

3. Створюємо порожній список.

4. Для введення елементів списку можна скористатись способом, розглянутим у теоретичному матеріалі.

5. Залишається знайти суму додатних елементів списку.

6. Програмна реалізація може бути такою.

```
A=[]
print("Введіть кількість елементів масиву.")
n=int(input("n="))
for i in range(n):
    print("A[",i,"]=", end=" ")
    A.append(int(input()))
print(A)
s=0
for i in range(n):
    if A[i]>0:
        s=s+A[i]
print('S = ', s)
```

7. Виконайте програму для кількох наборів даних.

8. Перегляньте результати виконання та при потребі змініть код програми. Перевірте, виконавши програму.

Завдання №2

Задано масив з *n* дійсних чисел. Визначте кількість елементів, значення яких дорівнюють цілому числу *c*. Метод *count()* використовувати забороняється. Збережіть програму у файлі з ім'ям «Завдання №2».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання №2».

2. Для розв'язування задачі використаємо тип даних *список.*

3. Сутність алгоритму розв'язування задачі полягає у послідовному порівнянні всіх елементів списку, починаючи з першого елемента, із заданим значенням. Якщо значення елемента масиву дорівнює заданому, то показник кількості *k* збільшується на одиницю.

4. Після створення виконайте програму для кількох наборів даних.

5. Перегляньте результати виконання та при потребі змініть коди програми. Перевірте, виконавши програму.

6. У випадку виникнення труднощів можете скористатися даним кодом програми:

```
a=[]
c=int(input('c='))
n=int(input('n='))
for i in range(n):
    print('a[', i,']=', end='')
    a.append(int(input()))
k=0
for i in range(n):
    if a[i]==c:
        k=k+1
print('k=', k)
```

Завдання №3

Задано масив з n дійсних чисел. Визначте значення максимального елемента та підрахуйте їх кількість. Функцію *max()* та метод *count()* використовувати забороняється. Збережіть програму у файлі з ім'ям «Завдання №3»

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання №3»

2. Спочатку визначимо значення максимального елемента в масиві. Для цього введемо змінну (наприклад MAX) та присвоїмо їй значення, що дорівнює першому елементу. Потім, здійснюючи перебір елементів масиву, будемо їх порівнювати з MAX, та якщо воно виявиться більшим, то MAX присвоїмо значення цього елемента і т. д.

3. Для підрахунку кількості максимальних елементів скористайтесь досвідом набутим під час виконання «Завдання №2».

4. Після створення програми виконайте її для кількох наборів даних.

5. Перегляньте результати виконання та при потребі змініть код програми.

6. У випадку виникнення труднощів можете орієнтуватись на код програми. Запропонований алгоритм не є оптимальним з точки зору часу виконання.

```
a=[]          # Створюємо порожній список
n=int(input('n='))  # Вводимо кількість елементів
for i in range(n):
    print('a[', i,']=', end='') # Вводимо елементи списку
    a.append(float(input()))
print(a)      # Виводимо список
MAX=a[1]
k=0
for i in range(n):
    if a[i]>MAX:        # Визначаємо максимальний елемент
        MAX=a[i]
for i in range(n):
    if a[i]==MAX:       # Визначаємо кільк. макс. елементів
        k=k+1
print('Максимальний елемент:', MAX)
print('Кількість максимальних:', k)
```

Завдання №4

Напишіть програму, під час виконання якої, буде створено та виведено на екран список цілих чисел з випадковою кількістю елементів *n* ($2 \le n \le 20$). Всі числа повинні відповідати умові: $1000 < A[i] < 1000$. Кількість елементів списку *n* також повинна бути виведена на екран. Фай для збереження з іменем «Завдання №4».

Технологія виконання

1. Створіть новий файл та збережіть його під ім'ям «Завдання №4».

2. Згенерувати *n* можна за допомогою функції *randint* з модуля *random*.

3. Створіть програму та виконайте її для кількох наборів даних.

4. Після аналізу вихідних даних, при потребі, змініть код програми. Перевірте виконавши програму.

5. У випадку виникнення труднощів скористайтеся кодом програми:

```
import random        # Під'єднуємо модуль "random"
n=random.randint(2, 20)   # Генеруємо кількість елементів списку
A=random.sample(range(-999, 1000), n) # Створюємо список
print('n=', n)      # Виводимо кількість елементів списку
print(A)      # Виводимо список
```

Завдання №5

Створіть функції користувача для створення масиву з *k* випадкових цілих чисел таких, що $d \le A[i] \le e$ та для знаходження середнього арифметичного елементів масиву. Згенеруйте три масиви з кількістю елементів n та знайдіть суму середніх арифметичних значень елементів кожного з них. Масиви також повинні бути виведені на екран. Файл для збереження із назвою «Завдання №5».

Технологія виконання

1. Створіть файл програми та збережіть його під ім'ям «Завдання №5».

2. Скориставшись досвідом, набутим під час виконання попередніх завдань, спочатку створюємо функцію для генерування масиву:

```
def arr_int(k, d, e):    # Функція для генерування масиву
    import random
    A=random.sample(range(d, e+1), k)
    return A
```

3. Потім функцію для знаходження середнього арифметичного елементів масиву:

```
def sa(B):  # Функція для знаходження сер. арифм. ел. масиву
    s=0
    for i in range(0, len(B)):
        s=s+B[i]
    p=s/len(B)
    return p
```

4. Доопрацюйте програмний код та запустіть програму на виконання.

5. Виконайте програму для кількох наборів даних.

6. Після аналізу вихідних даних, при потребі, зробіть корективи коду програми.

7. Якщо виникли труднощі, скористайтеся даним програмним кодом.

```
def arr_int(k, d, e):   # Функція для генерування масиву
    import random
    A=random.sample(range(d, e+1), k)
    return A
def sa(B):  # Функція для знаходження сер. арифм. ел. масиву
    s=0
    for i in range(0, len(B)):
        s=s+B[i]
    p=s/len(B)
    return p
n=int(input('n='))
a=int(input('a='))
b=int(input('b='))
C=arr_int(n, a, b)   # Генеруємо перший масив
sa1=sa(C)            # Знаходимо с. ар. ел. першого масиву
D=arr_int(n, a, b)   # Генеруємо другий масив
sa2=sa(D)            # Знаходимо с. ар. ел. другого масиву
E=arr_int(n, a, b)   # Генеруємо третій масив
sa3=sa(E)            # Знаходимо с. ар. ел. третього масиву
print(C)
print(sa1)
print(D)
print(sa2)
print(E)
print(sa3)
print('s=',sa1+sa2+sa3)
```

Завдання для самостійного виконання

1. Учні вели спостереження за температурою повітря на протязі *n* днів та заносили дані у таблицю. Складіть програму для знаходження середньої температури повітря для цього проміжку часу.

2. Знайти суму елементів цілочисельного масиву (таблиці) кратних числу 10.

3. У масиві визначте кількість елементів, значення яких менше числа *M*.

4. Створіть список з 7 елементів *(A[0], A[1], A[2], ..., A[6])* випадкових цілих чисел, таких, що *0 <A[i] < 1000* та виведіть його на екран.

5. Створіть програму, під час виконання якої, буде згенеровано та виведено на екран список цілих чисел з випадковою кількістю елементів *n* (*1 < n < 20*). Всі числа повинні належати іншому списку [7, 37, 25, 44, 15, 9, -7, -10, -12, 0]. Кількість елементів списку *n* також повинна бути виведена на екран.

6. Створіть функції користувача для генерування масиву із *k* випадкових цілих чисел таких, що $d \le A[i] \le e$ та для знаходження суми елементів масиву. Згенеруйте три масиви з кількістю елементів n та знайдіть середнє арифметичне сум елементів цих масивів. Масиви також повинні бути виведені на екран.

7. Згенеруйте 5 масивів кожний з яких містить *n* випадкових цілих чисел ($0 \le A[i] \le 1000$), виведіть їх на екран та визначте значення мінімального елементу в кожному з них. Функцію *min()* використовувати забороняється.

8. У масиві *a[0], a[1], a[2], ..., a[n-1]* визначте індекси та підрахуйте кількість максимальних елементів.

9. Створіть масив, що містить *n* (*n < 1000*) цілих чисел таких, що $a \le A[i] \le b$ (*0 < a < b, b < 1000000*). Підрахуйте кількість елементів менших за 5000 та замініть їх числом 5000.

10. Дано масив з *n* цілих чисел. Знайти суму всіх елементів масиву які не дорівнюють максимальному.

11. Дано масив з *n* цілих чисел. Виведіть всі його елементи з парними індексами.

12. Дано масив з *n* цілих чисел. Визначте, скільки в цьому масиві елементів, які більші двох своїх сусідів і виведіть кількість таких елементів. Крайні елементи списку не враховуються, оскільки у них мало сусідів.

13. Дано масив з *n* цілих чисел. Замінити всі найбільші елементи на найменший, а найменші на найбільший.

14. Дано масив з *n* цілих чисел. Знайти суму та кількість чисел, які більші за середнє арифметичне елементів масиву.

15. Дано масив з *n* цілих чисел. Знайти суму двох найменших елементів масиву.

16. Маємо *N* цілих чисел. Який найбільший добуток можна отримати, використавши тільки три з цих чисел?

3.6.3. Створення рядків

Рядки є впорядкованими послідовностями символів. Довжина рядка обмежена лише об'ємом оперативної пам'яті комп'ютера. Як і всі послідовності, рядки підтримують звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in).

Крім того, рядки належать до незмінних типів даних. Тому практично всі рядкові методи в якості значення повертають новий рядок. При використанні невеликих рядків це не призводить до будь-яких проблем, але при роботі з великими рядками можна зіткнутися з проблемою нестачі пам'яті [24].

Будь-яка послідовність символів, взята в лапки, в Python вважається рядком; при цьому рядки можуть бути взяті як в одиночні, так і в подвійні лапки:

```
>>> r1 = 'Це рядок'
>>> r2 = "Це теж рядок"
>>> type(r1)
<class 'str'>
>>> type(r2)
<class 'str'>
```

Це правило дозволяє використовувати внутрішні лапки та апострофи у рядках.

Створити рядок можна також вказавши набір символів між потроєнними апострофами або потроєнними лапками. Такі об'єкти можна розмістити на декількох рядках, а також одночасно використовувати лапки і апострофи. В інших випадках такі об'єкти еквівалентні рядкам в апострофах і лапках. Всі спеціальні символи в таких рядках інтерпретуються. Приклади:

```
>>> print('''Рядок 1
Рядок 2''')
Рядок 1
Рядок 2
>>> print("""Рядок 1
Рядок 2""")
Рядок 1
Рядок 2
```

Створити рядок можна також за допомогою функції str([<Об’єкт>[, <Кодування>[, <Обробка помилок>]]]). Якщо вказано тільки перший параметр, то функція повертає рядкове представлення будь-якого об’єкта. Якщо параметри не зазначені взагалі, то повертається порожній рядок.

```
>>> s = 55.5
>>> type(s)
<class 'float'>
>>> v = str(s)
>>> v
'55.5'
>>> type(v)
<class 'str'>
```

3.6.4. Зміна регістру символів у рядках

Одна з найпростіших операцій, що виконуються з рядками, дає змогу змінювати регістр символів. Погляньте на наступний фрагмент коду та спробуйте визначити, що у ньому відбувається:

```
>>> name = "ada lovelace"
>>> print(name.title())
Ada Lovelace
```

У цьому прикладі у змінній name зберігається рядок, що складається з букв нижнього регістру "ada lovelace". За ім’ям змінної в команді print() йде виклик методу title(). Метод являє собою дію, яку Python виконує з даними. Крапка (.) після name в конструкції name.title() наказує Python застосувати метод title() до змінної name. За ім’ям методу завжди повинна стояти пара круглих дужок, тому, що методам для виконання їх роботи часто потрібна додаткова інформація. Ця інформація вказується в дужках. Функції title() додаткова інформація не потрібна, тому в круглих дужках нічого немає.

Метод title() перетворює перший символ кожного слова у рядку до верхнього регістру, тоді як всі інші символи виводяться у нижньому регістрі. Наприклад, дана можливість може бути корисна, якщо у вашій програмі вхідні значення Ada, ADA та ada повинні розглядатися як одне й те ж саме ім'я, та всі вони повинні відображатися у вигляді Ada.

Для роботи з регістром також існують інші корисні методи. Наприклад, всі символи рядка можна перетворити до верхнього або нижнього регістру:

```
>>> name = "Ada Lovelace"
>>> print(name.upper())
ADA LOVELACE
>>> print(name.lower())
ada lovelace
```

Метод lower() особливо корисний для зберігання даних. Нерідко програміст не може розраховувати на те, що користувачі введуть всі дані з точним дотриманням регістру, тому рядки перед збереженням перетворюються до нижнього регістру. Потім, коли буде потрібно вивести інформацію, використовується регістр, найбільш підходящий для кожного рядка [24].

3.6.5. Конкатенація

Часто виникає необхідність у об'єднанні рядків. Уявіть, що ім'я та прізвище зберігаються у різних змінних та ви хочете об'єднати їх для виведення повного імені:

```
>>> first_name = "Ada"
>>> last_name = "Lovelace"
>>> full_name = first_name + " " + last_name
>>> print(full_name)
Ada Lovelace
```

Для об'єднання рядків у Python використовується знак «плюс» (+). У наведеному прикладі повне ім'я будується об'єднанням first_name, пробіла та last_name.

Такий спосіб об'єднання рядків називається конкатенацією. Ви можете використовувати конкатенацію для побудови складних повідомлень з інформацією, що зберігається у змінних.

Розглянемо приклад:

```
>>> first_name = "ada"
>>> last_name = "lovelace"
>>> full_name = first_name + " " + last_name
>>> print("Hello, " + full_name.title() + "!")
Hello, Ada Lovelace!
```

Повне ім’я використовується для виведення привітання, а метод title() забезпечує правильне форматування імені.

```
>>> first_name = "ada"
>>> last_name = "lovelace"
>>> full_name = first_name + " " + last_name
>>> message = "Hello, " + full_name.title() + "!"
>>> print(message)
Hello, Ada Lovelace!
```

Цей код також виводить повідомлення «Hello, Ada Lovelace!», Але збереження тексту повідомлення у змінній істотно спрощує завершальну команду виведення [24].

3.6.6. Табуляції та розриви рядків, екрановані послідовності

У програмуванні терміном «пропуск» (whitespace) називаються такі недруковані символи, як пробіли, табуляції та символи кінця рядка. Пропуски структурують текст, щоб користувачеві було зручніше читати його.

Для включення у текст позиції табуляції використовується комбінація символів \t.

```
>>> print("Python")
Python
>>> print("\tPython")
        Python
```

Розриви рядків додаються за допомогою комбінації символів \n:

```
>>> print("Мови програмування:\nPython\nC++\nJavaScript")
Мови програмування:
Python
C++
JavaScript
```

Табуляції та розриви рядків можуть поєднуватися у тексті. Скажімо, послідовність «\n\t» наказує Python почати текст з нового рядка, на початку якого розташовується табуляція.

Наступний приклад демонструє виведення одного повідомлення з розбивкою на чотири рядки:

```
>>> print("Мови програмування:\n\tPython\n\tC++\n\tJavaScript")
Мови програмування:
        Python
        C++
        JavaScript
```

3.6.7. Екрановані послідовності

Для повноцінного представлення структури тексту або його специфіки нам потрібні якісь особливі символи. Деякі символи повинні управляти курсором (ще називають «кареткою» – відгомін з минулого від друкарських машинок). Ще потрібні символи за допомогою яких можна виконувати якісь службові завдання, наприклад, позначати шістнадцятиричні або восьмеричні коди символів. Для таких цілей і існують екрановані послідовності [24].

Екрановані послідовності – це послідовності, які починаються з символу «\» за яким розташовано один або більше символів. Розглянемо ті, що найчастіше використовуються (табл. 3.6.1).

Таблиця 3.6.1

Послідовність	**Призначення**
\newline	Якщо після символу «\» відразу натиснути клавішу <Enter> то це дозволить продовжувати запис з нового рядка.
\\	Дозволяє записати символ зворотного слеша.
\’	Дозволяє записати один символ апострофа.
\”	Дозволяє записати один символ лапок.
\n	Перенесення рядка (новий рядок).
\t	Горизонтальний відступ зліва від початку рядка (горизонтальна табуляція).
\v	Вертикальний відступ зверху (вертикальна табуляція).
\other	Під other розуміється будь-яка інша послідовність символів. Залишається без змін зі збереженням у рядку символу «\».

Контрольні запитання

1. Що розуміють під рядками у мові Python?

2. Чим обмежена довжина рядка у Python?

3. Чи належать рядки до незмінних типів даних у мові програмування Python?

4. Що з переліченого підтримують рядки: звернення до елементу за індексом, отримання зрізу, конкатенацію (оператор +), повторення (оператор *), перевірку на входження (оператор in)?

5. Які вам відомі способи створення рядків у мові Python?

6. Як змінити регістр символів рядка у Python?

7. Що таке конкатенація?

8. За допомогою яких засобів можна видалити пропуски у мові програмування Python?

9. Які ви знаєте методи для роботи з рядками у Python?

10. Яке призначення функцій chr та ord?

11. Які методи в мові Python використовуються для здійснення пошуку та заміни у рядку?

Практичні завдання

Завдання №1

Створіть програму, яка запитує ім’я користувача, а потім його вітає, виводячи слово «Привіт», ім’я користувача та розділові знаки у наступному вигляді: «Привіт, Валерій!». Файл зберегти під ім’ям «Завдання №1».

Технологія виконання

1. Задача елементарна. Виводимо запрошення користувачу ввести власне ім’я, а потім, використавши операцію конкатенації, виводимо напис, як це вимагається в умові задачі.

2. Наведемо один з можливих варіантів програми:

```
a=input('Як Вас звати? ')
print('Привіт, '+a+'!')
```

3. Або так:

```
print('Привіт, '+input('Як Вас звати? ')+'!')
```

4. Результат виконання:

```
Як Вас звати? Валерій
Привіт, Валерій!
```

Завдання №2

Створіть програму для знаходження кількості одиниць у двійковому запису заданого числа. Файл зберегти під ім'ям «Завдання №2».

Технологія виконання

1. Для представлення цілих чисел у системах числення з основами 2, 8 та 16 у Python використовуються вбудовані функції *bin*, *oct* та *hex*, відповідно. Отже, щоб записати задане число у двійковій формі можна використати функцію *bin*, а метод count, поверне кількість символів «1».

2. Отримуємо один з варіантів програми:

```
a=int(input('Введіть натуральне число: '))
b=bin(a)
c=b.count('1')
print(b)
print(c)
```

3. Результат виконання:

```
Введіть натуральне число: 25
0b11001
3
```

4. Або такий варіант:

```
print(bin(int(input('Введіть натуральне число: '))).count('1'))
```

5. Результат виконання:

```
Введіть натуральне число: 25
3
```

Завдання №3

Непорожній рядок, який містить деяке слово, називається паліндромом, якщо це слово однаково читається як зліва направо, так і з права наліво

(наприклад: зараз, корок, ротатор, комок, біб, піп, дід, шалаш, кок, тут, око, вимив, вилив, вишив, вирив).

Створіть програму яка аналізує введене слово, що складається з символів одного регістру, та виводить повідомлення про те, чи є дане слово паліндромом. Файл зберегли під ім'ям «Завдання №3».

Технологія виконання

1. Щоб з'ясувати чи є введене слово паліндромом його потрібно «перевернути» та порівняти з початковим. Для перевертання слова використаємо операцію витягування зрізу, яка повертає зазначений фрагмент рядка.

2. Отримаємо програму:

```
sl=input('Введіть слово: ')
ls=sl[::-1]
if sl==ls:
    print('Це паліндром.')
else:
    print('Це не паліндром.')
```

3. Наведемо результати виконання для різних наборів даних:

```
Введіть слово: око
Це паліндром.
>>>
Введіть слово: весна
Це не паліндром.
```

Завдання №4

Створіть програму для знаходження кількості певних букв (вводить користувач) у даному тексті. Файл зберегти під назвою «Завдання №4».

Технологія виконання

1. Ідея розв'язання:

- вводимо букву або декілька (що будемо підраховувати);

- вводимо текст;

- за допомогою циклу for будемо перебирати букви у введеному тексті і якщо відповідна літера належить до тих, що потрібно підрахувати то лічильник збільшуватимемо на 1;

- виводимо кількість букв.

2. Програмна реалізація:

```
bukvi=input('Введіть букву або кілька: ')
text=input('Введіть текст: ')
k=0
for bukva in text:
    if bukva in bukvi:
        k=k+1
print(k)
```

3. Наведемо результати виконання для різних наборів даних:

```
Введіть букву або кілька: а
Введіть текст: Слава Україні!
3

Введіть букву або кілька: сво
Введіть текст: Душу й тіло ми положим за нашу свободу
7
```

4. Зауважимо, що програму можна використовувати для підрахунку й інших символів, наприклад, пробілів, крапок або цифр.

Завдання №5

Створити програму для з'ясування того, чи можна скласти слово з літер іншого слова. Файл зберегти під назвою «Завдання №5»

Технологія виконання

1. Ідея розв'язання полягає у наступному:

- вводимо початкове слово

- вводимо слово для оцінювання

- будемо переглядати літери слова по черзі і якщо не знаходимо певної букви у початковому слові, то це свідчитиме про те, що слово скласти неможливо

2. Наведемо один з варіантів програми:

```
cl_poch=input('Введіть перше слово: ')
cl=input('Введіть друге слово: ')
pok=True
for bukva in cl:
    if bukva not in cl_poch:
        pok=False
if pok:
    print('Слово скласти можна.')
else:
    print('Слово скласти не можна.')
```

3. Результати виконання для різних наборів даних:

```
Введіть перше слово: інформатика
Введіть друге слово: форма
Слово скласти можна.
>>>
Введіть перше слово: інформатика
Введіть друге слово: мова
Слово скласти не можна.
```

Завдання №6

У студента Васі є молодший брат Петро, який пішов до першого класу і почав вивчати арифметику. Додому у першому класі задали розв'язати багато прикладів на додавання та віднімання. Петя попросив Васю перевірити домашнє завдання. Побачивши дві сторінки написаних каракулями прикладів, Вася прийшов у жах від об'єму роботи і вирішив навчити Петю використовувати для самоперевірки комп'ютер. Для цього Васі потрібно написати програму, яка обчислювала б розв'язки потрібних арифметичних прикладів.

Вхідні дані

Один рядок, у якому можуть зустрічатись цифри та символи «+» і «-». Довжина рядка не перевищує 10000 символів, значення всіх чисел у ньому не перевищують 10000.

Вихідні дані

Вивести одне ціле число – результат обчислень.

Зберегти файл з ім'ям «Завдання №6».

Технологія виконання

1. Задача має дуже простий розв'язок, якщо використати функцію *eval*.

2. Лістинг програми:

```
a=input('Введіть вираз: ')
print(eval(a))
```

3. Результати виконання для одного з наборів даних:

```
Введіть вираз: 7+33-15
25
```

4. За допомогою цієї програми можна обчислювати значення виразів, що містять й інші дії. Наприклад:

```
Введіть вираз: (6+20)*5+2**3
138
```

Завдання №7

Створити програму для підрахунку цифр, великих та малих букв у введеному рядку. Зберегти файл під назвою «Завдання №7».

Технологія виконання

1. Задача розв'язується доволі просто за допомогою методів *isdigit()*, *isupper()*, та *islower()*.

2. Наведемо один з варіантів розв'язання:

```
a=input("Введіть текст: ")
v=0
m=0
c=0
for b in a:
    if b.isupper():
        v=v+1
    if b.islower():
        m=m+1
    if b.isdigit():
        c=c+1
print('Великих літер: ', v)
print('Малих літер: ', m)
print('Цифр: ', c)
```

3. Результат виконання [24].

```
Введіть текст: Завдання 1-17-16
Великих літер:  1
Малих літер:  7
Цифр:  5
```

Завдання для самостійного виконання

1. Створіть програму для знаходження кількості нулів у двійковому запису заданого числа.

2. Порахувати кількість букв «*а*» у введеному тексті.

3. Непорожній рядок, який містить деяке слово (або декілька), називається паліндромом, якщо це слово однаково читається як зліва направо, так і з права наліво (наприклад: зараз, корок, ротатор, комок, біб, піп, дід, шалаш, кок, тут, око вимив, вилив, вишив, вирив). Створіть програму яка аналізує введений рядок (врахувати, що букви можуть мати різні регістри), та виводить повідомлення про те, чи є дане слово паліндромом.

4. Як відомо, числа в двійковій системі записують за допомогою цифр 0 та 1. Ваше завдання – перевести число з двійкового подання в десяткове [24].

Розділ 4. ГЕЙМІФІКАЦІЯ. ІГРОВІ СИМУЛЯТОРИ

У сучасному суспільстві ігрові компоненти проникають до різних сфер життєдіяльності, підвищуючи при цьому рівень продуктивності, та найефективнішого впровадження технологія гейміфікації набуває саме в освітній діяльності.

У загальному розумінні технологія гейміфікації – це процес використання гри в неігрових ситуаціях, освітня ж гейміфікація передбачає застосування ігрових елементів, практик, технік у процесі здобуття знань, формування вмінь, навичок та ключових компетентностей [26].

Вивчення програмування – це поступовий процес, який має розпочинатись з вивчення алгоритмізації та основ програмування. Саме з допомогою ігор молоде покоління може здобути потрібні знання для «дорослого» кодингу. Розглянемо перелік онлайн вебресурсів та додатків, які дають змогу розпочати свій шлях в програмуванні через ігри:

- **Ігри Blockly** – це серія освітніх ігор, які навчають програмування. Вони призначені для дітей, які не мають досвіду роботи з комп'ютерним програмуванням. Після проходження ігор гравці будуть готові використовувати звичайні текстові мови.

- **Скажені Кролики Навчають Програмувати** – безкоштовний додаток для вивчення основ кодингу. У цій навчальній грі треба допомагати кроликам прибрати безлад на космічній станції. Для цього треба використовувати прості інструкції, цикли та алгоритми, що схожі на написання коду. Одна із задач – оптимізувати кількість рядків коду при досягненні цілі [8].

- **7 billion humans** – це сиквел успішного індіпроекту Human Resource Machine від розробника Tomorrow Corporation. Гра приваблива і самобутня, і головне, з гумором. Цього разу гравець має керувати не одним офісним працівником, а цілим загоном. Головна мета гри – познайомити вас із багатопоточністю процесів. Тут не потрібно писати код – все простіше: необхідно вибирати логічні блоки із запропонованих та вибудовувати їх у

правильну послідовність. Така простота припаде до душі не лише дорослим, а й дітям.

- **Kodu Game Lab** – візуальний конструктор, який дозволяє маленьким дітям створювати тривимірні ігри без знання основних мов програмування. Kodu доступний одночасно на ігрових приставках Xbox 360 і на персональних комп'ютерах. Крім того, Kodu Game Lab, на відміну від багатьох інших інструментів для розробки ігор, знайомить дітей з логікою програмування й засобами вирішення проблем, уникаючи складний синтаксис, що є ідеальним варіантом для тих, хто лише починає пізнавати ази створення ігор.

- **Minecraft education edition «Minecraft Hour of Code»** – це освітня версія однієї з найпопулярніших відеоігор в історії, яка перевершує всі платформи та приваблює гравців незалежно від віку, демографічної чи географічної ознаки, а одне з розширень «Minecraft Hour of Code» допоможе опанувати базові навички програмування як з допомогою блочної системи та і з допомогою класичного написання коду.

- **CodeCombat** – це багатокористувацька браузерна гра для навчання програмуванню. Починати грати можна без початкових навичок програмування. У грі команди коду подаються у вигляді заклять і дій починаючого чарівника або воїна, якого користувачу і треба «прокачати». Поступово користувач опановує наступні навички: базовий синтаксис, методи програмування, параметри, рядковий тип даних, цикли, змінні, оператор if, реляційні оператори, властивості об'єкта, обробка вводу, арифметика, нескінченні цикли While, оператор Break, масиви, порівняння рядків, знаходження Min/Max, ініціалізатор об'єкта, нескінченні цикли For, функції, візуальне програмування, типи даних тощо.

- **CodeMonkey** – це проста гра, в процесі якої діти можуть познайомитися з основами. Можна скористатися як 30-денною безплатною версією, так і оформити платну версію. Весь процес поділено на окремі рівні. У будь-який момент гри її можна призупинити (стан буде збережено) або повернутися на кілька кроків назад. Перед кожним рівнем подано підказки, тому впоратися із завданням зможуть навіть школярі молодших класів. Рівні розташовано в

порядку зростання складності. У перших 30 задачах CodeMonkey запроваджено такі поняття: об’єкти, функції, висловлювання, аргументи і простий цикл. У повному платному курсі CodeMonkey пропонує 400 завдань, що охоплюють складніші теми. Гра має три режими: вчитель, учень, домашній користувач. Вчитель може контролювати успіхи учнів, переглядати коди, які написали учні, аналізувати статистику досягнень класу тощо. Режим «учень» дає змогу приєднатися з обліковим записом до певного класу. Домашній користувач реєструється за стандартною процедурою. Під час гри треба керувати маленькою мавпочкою, яка ходить по ігровому полю і збирає банани [16].

Контрольні запитання

1. У чому полягає освітня гейміфікація?
2. Назвіть основні елементи гейміфікації?
3. Які ресурси гейміфікації для вивчення програмування Ви знаєте?

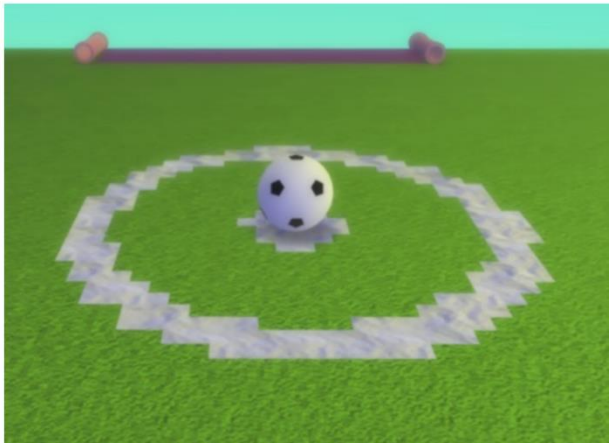
Практичні завдання

Завдання №1

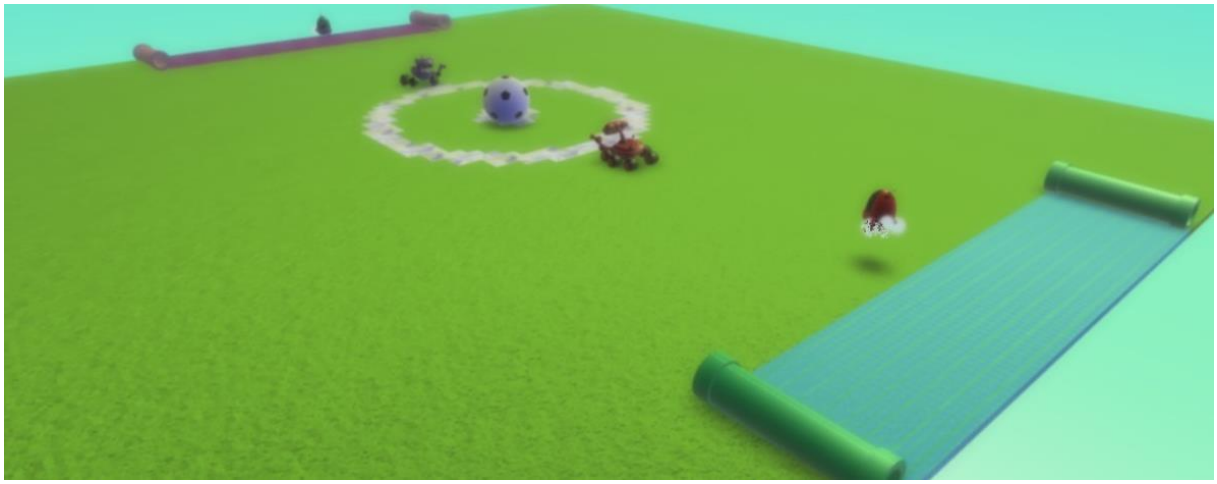
Створіть гру під назвою «Футбол». Створіть 2 команди по 2 гравця в кожній (1 воротар та 1 нападаючий). Керувати нападаючими мають гравці на клавіатурі, воротарі ж рухаються відповідно заданим їм умовам, а саме, рухатись до м’яча коли вони його побачать. Виграє команда, яка першою заб’є 5 голів у ворота суперника.

Технологія виконання

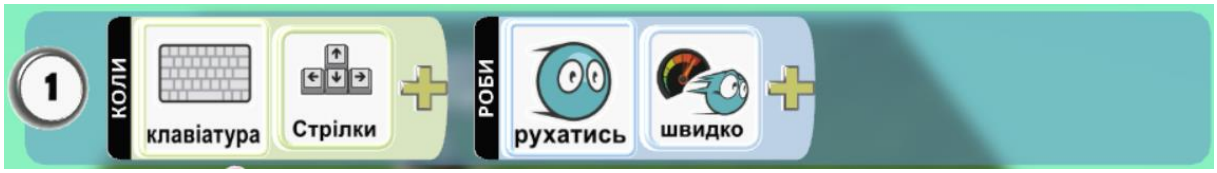
1. Створіть новий світ та намалюйте поле
2. Створіть ворота, використовуйте для цього об’єкт «труба», додайте труби з обох боків поля, які і будуть позначати межі воріт
3. Намалюйте центр та додайте м’яч



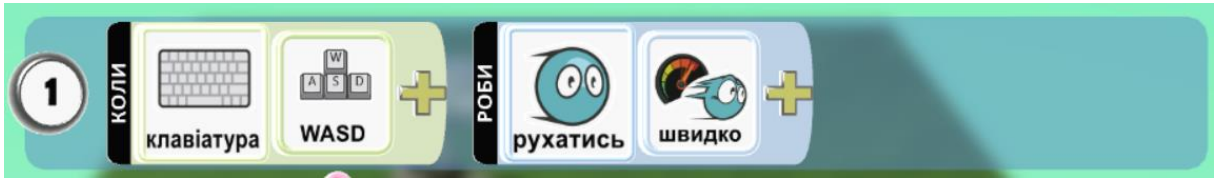
4. Додайте нападників та змініть їх колір, нехай це буде синій та червоний кольори або будь-які кольори на ваш вибір



5. Додайте програму для руху 1 нападаючого.



6. Додайте аналогічну програму для 2 нападаючого.



7. Додати програму для м’яча

8. Додайте вороторів, змініть їхній колір відповідно до кольору команди в яких вони гратимуть та запрограмуйте їх.

9. Додайте м’ячу програму перемоги

10. Додайте об’єкт «Куля», який виконуватиме роль коментатора. Запрограмуйте кулю.

11. Запустіть гру та перевірте її на працездатність.

Завдання для самостійного виконання

1. Ознайомтесь з ігровими сервісами, які було описано вище. Пройдіть один рівень Minecraft Education Edition «Minecraft Hour of Code» за допомогою блочної системи та ще один за допомогою коду.

2. Створіть свою гру використовуючи Kodu Game Lab в одному з жанрів, платформер, пригодницька, аркади, перегони.

3. Пройдіть перших **10 рівнів «**7 billion humans», «Скажені Кролики Навчають Програмувати», «Ігри Blockly».

Предметний покажчик

Список використаних джерел

1. Антонова О.П. Інформатика : Підруч. для 4 кл. закл. заг. серед. освіт. Тернопіль : Підруч. і посіб., 2021. 127 с.
2. Березовський В.Є., Ковальов Л.Є., Медведєва М.О. Чисельні методи з прикладами реалізації мовою Python : навчальний посібник. Умань : ВПЦ «Візаві», 2023. 88 с.
3. Бондарчук Ж.А. Збірка творчих завдань та вправ в середовищі програмування Scratch: методична розробка. Луцьк, 2017. 44 с.
4. Васильєв О.М. Програмування мовою Python. Тернопіль: Навчальна книга – Богдан, 2019. 504 с.
5. Жмурко О.І., Охріменко Т.О. Олімпіади з програмування. Прості задачі. Умань : Візаві, 2020. 298 с.
6. Івашко В.В. Основи програмування: конспект лекцій. Чернівці: Чернівецький національний університет імені Юрія Федьковича, 2021. 177 с.
7. Кащеєв Л.Б., Коваленко С.В., Коваленко С.М. Інформатика. Основи візуального програмування : навч. посіб. Харків : Веста, 2011. 192 с.
8. Ковтанюк М.С. Переваги використання мобільних ігрових симуляторів під час вивчення основ програмування. *Наука. Освіта. Молодь* : матеріали XVI Всеукр. наук. конф. студентів та молодих науковців, м. Умань, 11 травня 2023 р. Умань, 2023. С. 152–153.
9. Ковтанюк М.С. Переваги використання онлайн-середовища розробки Replit під час вивчення Програмування. *Комп'ютерні технології: інновації, проблеми, рішення* : тези доп. IV Всеукр. наук.-техн. конф., 18–20 листопада 2021. Житомир : Житомирська політехніка, 2021. С. 97–98.
10. Ковтанюк М.С., Тітова Л.О. Використання ігрових симуляторів під час вивчення програмування. *«Комп'ютерні технології: інновації, проблеми, рішення»* : Тези доп. IV Всеукр. науково-техн. конф., м. Житомир, 18–20 листоп. 2021 р. Житомир, 2021. С. 95–96.
11. Козолуп Є.В. Програмування в школі. Мова Python : навч. посібник. Суми, 2017. 82 с.

12. Костюченко А.О. Основи програмування мовою Python: навчальний посібник. Чернігів: ФОП Баликіна С.М., 2020. 180 с.

13. Лабораторний практикум з дисципліни «Алгоритмізація та програмування»: навч. посібник / Н.Б. Яворський та ін. Львів : Видавництво Львівської політехніки, 2018. 191 с.

14. Лабораторний практикум з програмування: навч. посібник / за ред. А.П. Власюка. Рівне: НУВГП, 2011. 495 с.

15. Матвійчук С.В., Жуковський С.С. Практикум програмування Python / C++ на e-olymp.com: зб. задач з рекомендаціями до їх розв'язання. Житомир: Вид-во ЖДУ, 2019. 235 с.

16. Медведєва М.О., Жмурко О.І., Криворучко І.І., Ковтанюк М.С. Використання ігрових онлайн-сервісів у процесі вивчення мов програмування. *Актуальні питання гуманітарних наук*. 2021. Т. 2, № 36. С. 248–255. URL: https://doi.org/10.24919/2308-4863/36-2-40.

17. Менський М. Scratch. Цікаві проєкти для дітей та дорослих. 2019. 31 с.

18. Мова асемблера. *www.wiki-data.uk-ua.nina.az*. URL: https://www.wiki-data.uk-ua.nina.az/Мова_асемблера.html.

19. Низькорівнева мова програмування. *www.wiki-data.uk-ua.nina.az*. URL: https://www.wiki-data.uk-ua.nina.az/Низькорівнева_мова_програмування.html.

20. Основи інформатики та технологій програмування: навчальний посібник / Рогоза М.Є. та ін. Луганськ: Вид-во СНУ ім. В. Даля, 2012. 568 с.

21. Основи програмування : методичні вказівки /уклад. А. Яковенко. Київ : НТУУ «КПІ ім. І. Сікорського», 2017. 87 с.

22. Пачесюк Л. Вивчаємо Scratch. 2016. 38 с.

23. Програмування числових методів мовою Python : підруч. / А.В. Анісімов та ін.; за ред. А.В. Анісімова. Київ : Видавничо-поліграфічний центр «Київський університет», 2014. 640 с.

24. Ракута В.М. Python у шкільному курсі інформатики. Основи програмування : навч. посібник. Чернігів, 2020. 160 с.

25. Руденко В., Жугастров О. Основи алгоритмізації і програмування мовою Python. 2-ге вид. Харків : Вид-во «Ранок», 2021. 192 с.

26. Тітова Л. Соціологічні аспекти освітньої гейміфікації. *Актуальні дослідження суспільних наук* : матеріали IX Всеукр. наук. конф., м. Умань, 23 берез. 2023 р. Умань, 2023. С. 28–29. URL: https://dspace.udpu.edu.ua/handle/123456789/15389.

27. Цифрові діти. Кодинг. Scratch 1 : навч. посіб. Київ : Лінгвіст, 2020. 40 с.

28. Учасники проєктів Вікімедіа. Байт-код – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Байт-код.

29. Учасники проєктів Вікімедіа. Скретч (мова програмування) – Вікіпедія. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Скретч_(мова_програмування)#Скретч_3.0.

30. Юрченко І.В., Сікора В.С. Програмування мовою Python: навч. посібник. Чернівці: Чернівецький національний університет, 2022. 104 с.

31. Яковенко А. Основи програмування. Python. Частина 1 : підручник. Київ : КПІ ім. Ігоря Сікорського, 2018. 195 с.

32. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1540 p.

33. Marji M. Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science, and Math. San Francisco, USA : No Starch Press, 2014. 288 p.

34. Matthes E. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019. 544 p.

35. Oleksandr B. PythonTask. *PythonTask*. URL: https://pythontask.pp.ua/.

36. Payne B. Teach your kids to code: A parent-friendly guide to Python programming. 2015. 308 p.

37. Pycharm це IDE від JetBrains для програмування на Python. *FoxmindEd*. URL: https://foxminded.ua/pycharm-tse/.

38. Scratch - Imagine, Program, Share. *Scratch - Imagine, Program, Share*. URL: https://scratch.mit.edu/

39. Stephenson B. The Python workbook: a brief introduction with exercises and solutions. Cham : Springer, 2014. 165 p.

40. Woodcock J. Coding Projects in Scratch: A Step-by-Step Visual Guide to Coding Your Own Animations, Games, Simulations, a. DK Children, 2019. 224 p.

ПРОГРАМУВАННЯ

Навчальний посібник

Укладачі Ковтанюк М. С., Тітова Л. О.

Видається в авторській редакції

Підписано до друку 13.11.2023 р. Формат 60х84/16.
Папір офсетний. Ум. друк. арк. 10,81
Тираж 100 прим. Замовлення № 2186

Видавничо-поліграфічний центр «Візаві»
20300, м. Умань, вул. Тищика, 18/19
Свідоцтво суб'єкта видавничої справи
ДК № 2521 від 08.06.2006.
тел. (093) 117-08-86, (067) 104-64-88
vizavi-print.jimdo.com
e-mail: vizavi008@gmail.com