ВІННИЦЬКИЙ ДЕРЖАВНИЙ ПЕДАГОГІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА КОЦЮБИНСЬКОГО

Факультет математики, фізики, комп'ютерних наук і технологій.
Кафедра Математики та інформатики

Д И П Л О М Н А Р О Б О Т А

на тему:

Методика вивчення основ програмування мовою Python майбутніми вчителями математики

Студента 2 курсу МСОІ групи
Освітньої програми Середня освіта. Інформатика, математика
Спеціальності 014.09 Середня освіта (Інформатика)
Галузі знань 01 Освіта/Педагогіка
Ступеня вищої освіти Магістр
Панченка Олександра Вікторовича

Науковий керівник Ковтонюк Г.М. Кандидат педагогічних наук, старший викладач кафедри математики та інформатики.

Розширена шкала _____

Кількість балів: _____ Оцінка: ECTS _____

Голова комісії _____ _____
(підпис) (ініціали, прізвище)

Члени комісії _____ _____
(підпис) (ініціали, прізвище)

_____ _____
(підпис) (ініціали, прізвище)

_____ _____
(підпис) (ініціали, прізвище)

_____ _____
(підпис) (ініціали, прізвище

м. Вінниця – 2019 рік

Анотація

Мета дослідження - розробка методики вивчення мови програмування Python майбутніми вчителями математики.

Об’єктом дослідження є процес вивчення мови програмування Python. Предметом дослідження є організаційно-методичні умови ефективного вивчення цієї мови програмування майбутніми вчителями математики. У роботі розроблено навчально-методичний комплекс для вивчення мови програмування Python майбутніми вчителями математики. Створено блог на якому розміщено всю інформацію по курсу та зроблена експериментальна перевірка ефективності розробленої методики та аналіз результатів дослідження.

Annotation

The purpose of the study is to develop a methodology for learning Python programming language by future math teachers.

The object of the study is the process of learning the Python programming language. The subject of the study is organizational and methodological conditions for the effective learning of this programming language by future mathematics teachers. In the work the educational-methodical complex for studying of the Python programming language by the future teachers of mathematics is developed. A blog has been created which provides all the information on the course and experimental verification of the effectiveness of the developed methodology and analysis of the research results.

Зміст

Вступ

З упевненістю можна сказати, що кожне наступне покоління є більш обізнаним в сфері інтернет технологій та програмування. Одним з факторів, що сприяють такому стану речей ми вважаємо інформатизацію суспільства.

Адаптація до змін в сучасному суспільстві супроводжується змінами в системі освіти. Одним з позитивних змін для вчителів залучення мови Python в процес викладання.

Вивчення мови програмування Python майбутніми вчителями математики може мати два цільових аспекти: перший – розвиває аспект, під яким розуміється розвиток алгоритмічного (ще говорять – операційного) мислення вчителів; другий – програмістський аспект. Складання методики для вивчення мови програмування Python вчителями математики – це важливе питання, яке постає в сфері освіти. Якщо спочатку, у цьому вивченні головним був програмістський аспект, який робив вивчення мов програмування вчителями інших спеціальностей другорядним питанням, то сьогодні перевага віддається першому аспекті. Саме така зміна зумовлює **актуальність** нашої роботи.

Питання про вивчення програмування вчителями інших спеціальностей залишається дискусійним. У різних версіях обов’язкового мінімуму це питання вирішувалося по-різному. Тут знову можна говорити про два цільових аспекти, з якими пов’язане вивчення програмування майбутніми вчителями математики. Перший аспект пов’язаний з посиленням фундаментального компоненту курсу математики. Спочатку учителям дається уявлення про те, що таке мови програмування, що представляє собою програма на мовах програмування високого рівня, як створюється програма в середовищі сучасної системи програмування. Отримавши уявлення про мову машинних команд на матеріалі навчальних комп’ютерів і про мови високого рівня, вчителі будуть усвідомлено сприймати поняття «трансляція». Другий аспект носить профорієнтаційний характер.

Тому **метою** дослідження є розробка методики вивчення мови програмування Python майбутніми вчителями математики.

Об'єктом дослідження є процес вивчення мови програмування Python.

Предметом дослідження є організаційно-методичні умови ефективного вивчення вивчення цієї мови програмування майбутніми вчителями математики.

Відповідно до мети були поставлені наступні **завдання**:

- вивчити наукову, методичну, історичну, спеціалізовану літературу з програмування та методики його вивчення; зробити аналіз літератури;
- провести аналіз існуючих авторських підходів до вивчення програмування майбутніми вчителями;
- зробити порівняльний аналіз мов програмування, які вивчаються майбутніми вчителями математики і виділити переваги мови програмування Python;
- розробити методику вивчення мови програмування Python майбутніми вчителями математики;
- провести експериментальну перевірку ефективності розробленої методики.

Використовувані **методи** дослідження:

1. Теоретичні:

аналіз наукової, методичної, історичної, спеціалізованої літератури з теми дослідження.

2. Емпіричні:

- спостереження;

- тестування;

- педагогічний експеримент, його аналіз.

Практична значимість дослідження полягає в тому, що розроблена методика з вивчення програмування мовою Python може використовуватись при вивченні програмування студентами математичних спеціальностей. Вони

можуть послужити основою розробки навчальних і методичних посібників для підготовки майбутніх вчителів математики.

Структура роботи. Робота складається із 10 рисунків, 1 таблиці 4 діаграм, вступу, двох розділів, висновку, списку використаних джерел, що містить 28 позицій та 3 Додатків.

РОЗДІЛ 1. ОСОБЛИВОСТІ МОВИ ПРОГРАМУВАННЯ PYTHON

1.1.Мови програмування та їх класифікація. Парадигми програмування

Мова програмування – це система позначень, що служить для точного опису програм або алгоритмів для ЕОМ. Мови програмування є штучними мовами. Від природних мов вони відрізняються обмеженим числом "слів" і дуже строгими правилами запису команд (операторів). Тому при застосуванні їх за призначенням вони не допускають вільного тлумачення виразів, характерного для природної мови.

Можна сформулювати ряд вимог до мов програмування і класифікувати мови за їх особливостям.

Основні вимоги, що пред’являються до мов програмування:

- наочність – використання в мові по можливості вже існуючих символів, добре відомих і зрозумілих як програмістам, так і користувачам ЕОМ;
- єдність – використання одних і тих же символів для позначення одних і тих же або споріднених понять в різних частинах алгоритму; кількість цих символів має бути по можливості мінімальною;
- гнучкість – можливість щодо зручного, нескладного опису поширених прийомів математичних обчислень за допомогою наявного в мові обмеженого набору образотворчих засобів;
- модульність - можливість опису складних алгоритмів у вигляді сукупності простих модулів, які можуть бути складені окремо і використані в різних складних алгоритмах;
- однозначність – недвозначність запису будь-якого алгоритму. Відсутність її могло б привести до неправильних відповідей при вирішенні завдань.

В даний час в світі існує кілька сотень реально використовуваних мов програмування. Для кожної є своя сфера застосування.

Будь-який алгоритм – це послідовність приписів, виконавши які можна за певне число кроків перейти від вихідних даних до результату. Залежно від

ступеня деталізації приписів зазвичай визначається рівень мови програмування – чим менша деталізація, тим вище рівень мови.

За цим критерієм можна виділити такі рівні мов програмування:

- машинні;

- машинно-орієнтовані (асемблери);

- машинно-незалежні (мови високого рівня).

Машинні мови і машинно-орієнтовані мови – це мови низького рівня, що вимагають вказівки дрібних деталей процесу обробки даних. Мови ж високого рівня імітують природні мови, використовуючи деякі слова розмовної мови і загальноприйняті математичні символи. Ці мови більш зручні для людини.

Різні типи процесорів мають різні набори команд. Якщо мова програмування орієнтована на конкретний тип процесора і враховує його особливості, то він називається мовою програмування низького рівня. В даному випадку "низький рівень" не означає "поганий". Мається на увазі, що оператори мови близькі до машинного коду і орієнтовані на конкретні команди процесора.

При програмуванні на машинній мові програміст може тримати під своїм контролем кожну команду, використовувати всі можливості наявних машинних операцій. Але процес написання програми на машинній мові дуже трудомісткий і виснажливий. Програма виходить громіздкою, її важко налагоджувати, змінювати і розвивати.

Тому в разі, коли потрібно мати ефективну програму, в максимальному ступені, враховується специфіка конкретного комп’ютера, замість машинних мов використовують близькі до них машинно-орієнтовані мови (асемблери).

Мова асемблера – це машинно-залежна мова низького рівня, в якій короткі мнемонічні імена відповідають окремим машинним командам. Використовується для подання в зрозумілій формі програм, записаних в машинному коді.

Мова асемблера дозволяє програмісту користуватися текстовими мнемонічними (тобто тими, які легко запам'ятовуються людиною) кодами, на свій розсуд присвоювати символічні імена регістрів комп'ютера і пам'яті, а також задавати зручні для себе способи адресації. Крім того, вона дозволяє використовувати різні системи числення (наприклад, десяткову або шістнадцяткову) для представлення числових констант, використовувати в програмі коментарі та ін.

За допомогою мов низького рівня створюються дуже ефективні і компактні програми, оскільки розробник отримує доступ до всіх можливостей процесора. З іншого боку, при цьому потрібно дуже добре розуміти будову комп'ютера, ускладняється налагодження великих додатків, а остаточна програма не може бути перенесена на комп'ютер з іншим типом процесора. Подібні мови зазвичай застосовують для написання невеликих системних додатків, драйверів пристроїв, модулів стикування з нестандартним обладнанням, коли найважливішими вимогами стають компактність, швидкодія і можливість прямого доступу до апаратних ресурсів. У деяких областях, наприклад в машинній графіці, на мові асемблера пишуться бібліотеки, ефективно реалізують алгоритми обробки зображень, що вимагають інтенсивних обчислень.

Таким чином, програми, написані на мові асемблера, вимагають значно меншого обсягу пам'яті і часу виконання. Знання програмістом мови асемблера і машинного коду дає йому розуміння архітектури машини. Незважаючи на те, що більшість фахівців в області програмного забезпечення розробляють програми на мовах високого рівня, найбільш потужне і ефективне програмне забезпечення повністю або частково написано на мові асемблера.

Мови високого рівня були розроблені для того, щоб звільнити програміста від урахування технічних особливостей конкретних комп'ютерів, їх архітектури. Рівень мови характеризується ступенем його близькості до природної, людської мови. Машинний мова не схожа на людську, вона вкрай бідна в своїх образотворчих засобах. Засоби запису програм на мовах високого

рівня більш виразні і звичні для людини. Наприклад, алгоритм обчислення по складній формулі не розбивається на окремі операції, а записується компактно у вигляді одного виразу з використанням звичної математичної символіки. Скласти свою або зрозуміти чужу програму такою мовою набагато простіше.

Важливою перевагою мов високого рівня є їх універсальність, незалежність від ЕОМ. Програма, написана такою мовою, може виконуватися на різних машинах. Упорядник програми не потрібно знати систему команд ЕОМ, на якій він передбачає проводити обчислення. При переході на іншу ЕОМ програма не вимагає переробки. Такі мови – це не тільки засіб спілкування людини з машиною, але і людей між собою. Програма, написана на мові високого рівня, легко може бути зрозуміла будь-яким фахівцем, який знає мову і характер завдання.

Таким чином, можна сформулювати основні переваги мов високого рівня перед машинними:

- алфавіт мови високого рівня значно ширше алфавіту машинної мови, що істотно підвищує наочність тексту програми;
- набір операцій, допустимих для використання, не залежить від набору машинних операцій, а вибирається з міркувань зручності формулювання алгоритмів розв’язання задач певного класу;
- формат пропозицій досить гнучкий і зручний для використання, що дозволяє за допомогою однієї пропозиції задати досить змістовний етап обробки даних;
- необхідні операції задаються за допомогою загальноприйнятих математичних позначень;
- даними в мовах високого рівня присвоюються індивідуальні імена, обрані програмістом;
- в мові може бути передбачений значно ширший набір типів даних в порівнянні з набором машинних типів даних.

Таким чином, мови високого рівня в значній мірі є машинно-незалежними. Вони полегшують роботу програміста і підвищують надійність створюваних програм.

Основні компоненти алгоритмічної мови:

- алфавіт;
- синтаксис;
- семантика.

Алфавіт – це фіксований для даної мови набір основних символів, тобто "Букв алфавіту", з яких повинен складатися будь-який текст на цій мові – ніякі інші символи в тексті не допускаються.

Синтаксис – це правила побудови фраз, що дозволяють визначити, правильно чи неправильно написана та чи інша фраза. Точніше кажучи, синтаксис мови являє собою набір правил, що встановлюють, які комбінації символів є осмисленими пропозиціями на цій мові.

Семантика визначає смислове значення пропозицій мови. Будучи системою правил тлумачення окремих мовних конструкцій, семантика встановлює, які послідовності дій описуються тими чи іншими фразами мови і, в кінцевому підсумку, який алгоритм визначений за цим текстом на алгоритмічній мові.

Мови високого рівня діляться на:

- процедурні;
- логічні;
- об’єктно-орієнтовані.

Процедурні мови призначені для однозначного опису алгоритмів. При вирішенні завдання процедурні мови вимагають в тій чи іншій формі явного запису процедуру її рішення.

Першим кроком у розвитку процедурних мов програмування була поява проблемно-орієнтованих мов. У цій назві відображений і той факт, що при їх

розробці йдуть не від «машини», а «від завдання»: в мові прагнуть максимально повно врахувати специфіку класу задач, для вирішення яких його передбачається використовувати. Наприклад, для багатьох науково-технічних завдань характерні великі розрахунки за складними формулами, тому в орієнтованих на такі завдання мовах вводять зручні засоби їх запису. Використання понять, термінів, символів, звичних для фахівців відповідної галузі знань, полегшує їм вивчення мови, спрощує процес складання і налагодження програми.

Різноманітність класів задач привело до того, що на сьогоднішній день розроблено декілька сотень алгоритмічних мов. Правда, широке поширення і міжнародне визнання отримали лише 10-15 мов. Серед них в першу чергу слід відзначити: Fortran і Algol – мови, призначені для вирішення науково-технічних завдань, Cobol – для вирішення економічних завдань, Basic – для вирішення невеликих обчислювальних задач в діалоговому режимі. В принципі кожна з цих мов може використовуватись для вирішення завдань не свого класу. Однак, як правило, застосування виявляється не зручним.

У той же час в середині 60-х років почали розробляти алгоритмічні мови широкої орієнтації – універсальні мови. Зазвичай вони будувалися за принципом об’єднання можливостей вузько-орієнтованих мов. Серед них найбільш відомі PL / 1, Pascal, C, C +, Modula, Ada. Однак, як будь-який універсальний засіб, такі широко-орієнтовані мови в багатьох конкретних випадках виявляються менш ефективними.

Логічні мови – (Prolog, Lisp, Mercury, KLO і ін.) орієнтовані не на запис алгоритму розв’язання задачі, а на систематичний і формалізований опис задачі з тим, щоб рішення випливало з складеного опису. У цих мовах вказується що дано і що потрібно отримати. При цьому пошук рішення задачі покладається безпосередньо на ЕОМ.

Керівною ідеєю об’єктно-орієнтованих мов (Object Pascal, C++, Java, Objective Caml. та ін.) є прагнення зв’язати дані з обробними процедурами цих даних в єдине ціле – об'єкт.

Об’єктно-орієнтований підхід використовує наступні базові поняття:

- об’єкт;
- властивість об’єкта;
- метод обробки;
- подія;
- клас об’єктів.

Об'єкт – сукупність властивостей (параметрів) певних сутностей і методів їх обробки (програмних засобів).

Властивість – це характеристика об’єкта та його параметрів. Всі об’єкти наділені певними властивостями, сукупність яких виділяють (визначають) об’єкт.

Метод – це набір дій над об’єктом або його властивостями.

Подія – це характеристика зміни стану об’єкта.

Клас – це сукупність об’єктів, що характеризуються спільністю застосовуваних до них методів обробки або властивостей.

Існують різні об’єктно-орієнтовані технології, які забезпечують виконання найважливіших принципів об’єктного підходу:

- Інкапсуляція;
- Успадкування.

Під інкапсуляцією розуміється приховування полів об’єкта з метою забезпечення доступу до них тільки за допомогою методів класу (тобто приховування деталей, несуттєвих для використання об’єкта). Інкапсуляція (об’єднання) означає поєднання даних і алгоритмів їх обробки, в результаті чого і дані, і процедури багато в чому втрачають самостійне значення.

Клас може мати утворені від нього підкласи. При побудові підкласів здійснюється успадкування даних і методів обробки об’єктів вихідного класу.

Фактично об’єктно-орієнтоване програмування можна розглядати як модульне програмування нового рівня, коли замість багато в чому випадкового,

механічного об’єднання процедур і даних акцент робиться на їх смисловий зв’язок.

Програма на об’єктно-орієнтованій мові, розв’язуючи деяку задачу, по суті, описує частину світу, що відноситься до цього завдання. Опис дійсності в формі системи взаємодіючих об’єктів природніша, ніж в формі взаємодіючих процедур.

1.2 Загальні відомості про мову Python.

Python – це універсальна сучасна МП високого рівня, до переваг якої відносять високу продуктивність програмних рішень і структурований код, що легко читається. Синтаксис Python максимально полегшений, що дозволяє вивчити його за порівняно короткий час. Ядро має дуже зручну структуру, а широкий перелік вбудованих бібліотек дозволяє застосовувати значний набір корисних функцій і можливостей. МП може використовуватися для написання прикладних програм, а також розробки WEB-сервісів.

Python може підтримувати широкий перелік стилів розробки додатків, в тому числі, дуже зручний для роботи з ООП і функціонального програмування.

Один з найпопулярніших інтерпретаторів мови – CPython, написаний на Сі. Поширюється це середовище розробки безкоштовно по вільній ліцензії. Інтерпретатор підтримує більшість популярних платформ.

Python активно розвивається, адже раз в 2 роки виходять оновлення. Важливою особливістю мови є відсутність таких стандартів кодування як ANSI, ISO і деяких інших, вони працюють завдяки інтерпретаторові.

1.2.1 Історія створення

Історія мови програмування Python почалася в кінці 1980-х. Гвідо ван Россум задумав Python в 1980-х роках, а приступив до його створення в грудні 1989 року в Центрі математики та інформатики в Нідерландах. Мова Python була задумана як нащадок мови програмування ABC, здатної до обробки виключень і взаємодії з операційною системою Амеба. Ван Россум є основним автором Python і продовжував виконувати центральну роль в ухваленні рішень щодо розвитку мови аж до 12 липня 2018 року.

У лютому 1991 року Гвідо ван Россум опублікував код Python, позначений версією 0.9.0, на alt.sources. На цій стадії в ньому вже були присутні класи з успадкуванням, обробка виключень, функції та основні типи даних: list, dict, str і т. д. Також в цьому початковому релізі були модулі, запозичені з Modula-3. Ван Россум описував модуль як «один з головних елементів в програмуванні на Python». Модель обробки виключень в Python теж була схожа

на Modula-3 з додаванням оператора else. У 1994 році з ростом числа користувачів сформувалася група comp.lang.python – основний форум Python.

Версія Python 2.0 була випущена 16 жовтня 2000 року і включала в себе багато нових великих функцій – таких як повний збирач сміття і підтримка Unicode. Однак найбільш важливою з усіх змін була зміна самого процесу розвитку мови і перехід на більш прозорий процес його створення.

Перша зворотньо-несумісна версія Python 3.0 була випущена 3 грудня 2008 року після тривалого періоду тестування. Багато її функцій були перенесені в сумісні Python 2.6 і Python 2.7.

1.2.2 Основні типи і структури даних

Типи даних

В Python типи даних можна розділити на вбудовані в інтерпретатор (built-in) і не вбудовані, які можна використовувати при імпортуванні відповідних модулів.

До основних вбудованих типів відносяться:

- None (невизначене значення змінної);
- Логічні змінні (Boolean Type);
- Числа (Numeric Type):

  int – ціле число;

  float – число з плаваючою точкою;

  complex – комплексне число;

- Списки (Sequence Type):

  List – список;

  tuple – кортеж;

  range – діапазон;

- Рядки (Text Sequence Type):

Str;

- Бінарні списки (Binary Sequence Types)

bytes – байти;

bytearray – масиви байт;

memoryview – спеціальні об’єкти для доступу до внутрішніх даних об’єкта через protocol buffer;

- Множини (Set Types):

set – множина;

frozenset – незмінна множина;

- Словники (Mapping Types):

dict – словник.

Модель даних

Розглянемо як створюються об’єкти в пам’яті, їх пристрій, процес оголошення нових змінних і роботу операції присвоювання.

Для того, щоб оголосити і відразу форматувати змінну необхідно написати її ім’я, потім поставити знак рівності і значення, з яким ця змінна буде створена. Наприклад рядок:

```
b = 5
```

оголошує змінну b і привласнює їй значення 5.

Цілочисельне значення 5 в рамках мови Python по суті своїй є об’єктом. Об’єкт, в даному випадку – це абстракція для представлення даних, дані – це числа, списки, рядки і т. д. При цьому, під даними слід розуміти як безпосередньо самі об’єкти, так і відносини між ними. Кожен об’єкт має три атрибути – це ідентифікатор, значення і тип. Ідентифікатор – це унікальна ознака об’єкта, що дозволяє відрізняти об’єкти один від одного, а значення –

безпосередньо інформація, що зберігається в пам’яті, якою управляє інтерпретатор.

При ініціалізації змінної, на рівні інтерпретатора, відбувається наступне:

- створюється цілочисельний об'єкт 5 (можна уявити, що в цей момент створюється осередок і 5 кладеться в цей осередок);
- даний об'єкт має певний ідентифікатор, значення: 5, і тип: ціле число;
- за допомогою оператора "=" створюється посилання між змінною b і цілочисельним об’єктом 5 (змінна b посилається на об’єкт 5).

Ім'я змінної не повинно збігатися з ключовими словами інтерпретатора Python.

В Python існують змінювані і незмінні типи.

До незмінних (immutable) типів відносяться: цілі числа (int), числа з плаваючою точкою (float), комплексні числа (complex), логічні змінні (bool), кортежі (tuple), рядки (str) і незмінні множини (frozen set).

До змінних (mutable) типів відносяться: списки (list), множини (set), словники (dict).

Як вже було сказано раніше, при створенні змінної, спочатку створюється об’єкт, який має унікальний ідентифікатор, тип і значення, після цього змінна може посилатися на створений об’єкт.

Незмінюваність типу даних означає, що створений об’єкт більше не змінюється.

Типи структури даних

Розглянемо структури даних, які містяться в останній версії Python 3.0 та є основними для інших версій. До них відносять:

- списки (list);

- кортежі (turple);

- словники (dictionary);

- множини (set);

- послідовності.

Список (list) в Python являє собою впорядковану колекцію елементів, тобто зберігає послідовність елементів в списку. В Python елементи списку розділяються комами.

Такий список елементів повинен бути укладений у квадратні дужки. Після того, як список створено можливо додавати, видаляти і виконувати пошук за елементами в ньому.

Оскільки ми можемо додавати і видаляти дані в списках – то вони називаються змінними (mutable) типами даних.

Кортежі використовуються для зберігання декількох об’єктів. По суті кортежі схожі зі списками, з тією різницею що вони не надають такої функціональності, як списки. Головна їхня відмінність та, що кортежі є незмінним (immutable) типом даних.

Словник – це щось на кшталт «записної книжки» – адреса людини (знаючи тільки її ім’я): у словнику ключ (key) асоціюється зі значенням (value). Всі ключі в одному словнику повинні бути унікальними. Крім того для ключів потрібно використовувати незмінний тип даних (наприклад – рядки), але при цьому – додавати будь-який тип даних в значення.

Значення в словнику вказуються за допомогою нотації dict = {key1: value1, key2: valu2}. Всі ключі в словнику зберігаються в неврегульованому стані. Словники є об’єктами класу dict.

Списки, кортежі і рядки є прикладами послідовностей. Головною відмінною можливістю послідовностей є перевірка членства (membership test), тобто застосування виразів in і not in, а так само використання операцій індексування (indexing operations), які дозволяють отримувати певні елементи з послідовності.

Три типи послідовностей, згаданих вище – списки, кортежі і рядки, так само мають можливість застосування слайсинга (slicing), яка дозволяє отримати частину послідовності.

Множинності є неврегульованими колекціями простих об’єктів, і використовуються в тих випадках, коли присутність об’єкта в колекції важливіша, ніж порядок або кількість входжень цього елемента в одній колекції.

Використовуючи множинності можливо використовувати перевірки членства, додавати і видаляти елементи, об’єднувати їх, виконувати перевірку на приналежність іншій множинності і так далі. Елементом множини може бути будь-який незмінний тип даних – числа, рядки, кортежі.

1.2.3 Синтаксис і семантика

Синтаксис мови Python, як і сама мова, дуже простий. Охарактеризуємо його наступним чином:

- Кінець рядка є кінцем інструкції (крапки з комою не потрібно).
- Вкладені інструкції об’єднуються в блоки по величині відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. Не варто також забувати про читаність коду. Відступ в 1 пробіл, наприклад, не найкраще рішення. Рекомендується використовувати 4 пробіли (або знак табуляції).
- Вкладені інструкції в Python записуються відповідно до одних і тих же шаблонів, коли основна інструкція завершується двокрапкою, слідом за якою розташовується вкладений блок коду, зазвичай з відступом під рядком основної інструкції.

Іноді можливо записати кілька інструкцій в одному рядку, розділяючи їх крапкою з комою:

```
a = 1; b = 2; print(a, b)
```

Проте не рекомендується робити це занадто часто. Слід пам’ятати про зручність читання.

Дозволяється записувати одну інструкцію в декількох рядках. Досить її укласти в пару круглих, квадратних або фігурних дужок:

```
        if (a == 1 and b == 2 and

  c == 3 and d == 4):

    print('spam' * 3)
```

Тіло зіставленої інструкції може розт ашовуватися в тому ж рядку, що і тіло основної, якщо тіло зіставленої інструкції не містить складових інструкцій. Наприклад:

```
        if x > y: print(x)
```

1.3 Області застосування мови Python

Python – це не просто мова програмування. Це цілий світ зі своїми можливостями, важкими завданнями і способами їх рішень. Новачкові, який тільки почав знайомство з мовою, досить важко усвідомити, в яких областях можуть стати в нагоді його знання.

Насправді, вибір досить великий. Python з кожним днем все активніше завойовує ринок, і на сьогоднішній день він займає одну з лідируючих позицій серед всіх інших мов, змагаючись за першість з «монолітами» індустрії.

Звичайно, Пітон ніколи не зможе замінити низькорівневі C і C ++, адже саме вони здатні практично повністю контролювати процесор, не займе місце Java, котрий призначений для розробки складних застосувань. Також, Python можна назвати аналогом JavaScript, який підтримується величезною часткою сайтів.

Мова Python – проста і універсальна, тому може застосовуватися для роботи в багатьох сферах.

Web-розробка

На Python можна робити весь backend інтернет-ресурсу, який буде виконуватися на сервері. Робиться це за допомогою спеціальних фреймворків (Django і Flask), написаних на цій мові. З їх допомогою спрощується процес обробки адрес, звернення до баз даних та створення HTML, котрі відображаються на призначених для користувача сторінках.

Сьогодні сторонніми розробниками написано велику кількість додаткового інструментарію, направлених на реалізацію мережевих додатків. Наприклад, інструмент HTMLGen дозволяє створювати готові класи під сторінку на HTML, використовуючи для цього мову Python. А пакет mod_python полегшує запуск сценаріїв Apache, забезпечуючи при цьому стабільну роботу шаблонів Python Server Pages.

Графічний інтерфейс

Якщо говорити про візуальні складові в сфері IT, то і тут Python може показати себе як цілком ефективний інструмент, який вирішує масу завдань. Створюючи сучасні графічні інтерфейси на Python, можна легко підлаштуватися під стилістику ОС, в середовищі якої створюється додаток. Спеціально для цих цілей були створені додаткові бібліотеки для побудови інтерфейсу – PythonCard і Dabo, що полегшують процес роботи.

Бази даних

Розробники сучасної версії Python створили максимально простий і зрозумілий доступ практично до будь-яких баз даних. Так, в робочому середовищі мови знаходиться програмний інтерфейс, який дозволяє користуватися базами прямо зі сценарію за допомогою запитів SQL. Також, код, написаний на Python, може з мінімальними доробками використовуватися для баз даних MySQL і Oracle.

Системне програмування

Ще одна монетка в скарбничку можливостей Python – це інтерфейси мови, які дозволяють управляти службами операційних систем Windows, Linux і ін. Завдяки цьому, Пітон відкриває масу можливостей для створення портативних програм. Не секрет, що ця мова застосовується для створення програмного забезпечення, що використовуються системними адміністраторами. Таким чином, Python прискорює пошук і відкриття файлів, запуск програм, полегшує обчислення.

Складні обчислювальні процеси

У цій сфері Python може конкурувати із FORTRAN або C ++. Спеціальне розширення NumPy, написане для математичних розрахунків, прекрасно функціонує з масивами, інтерфейсами рівнянь і іншими даними. Як тільки розширення встановлюється на комп’ютер, Python без проблем проходить інтеграцію з бібліотеками формул.

Але NumPy призначений не тільки для обчислень. Крім свого основного завдання, з його допомогою можна створювати анімовані елементи і

промальовувати об’єкти в середовищі 3D, виробляючи при цьому паралельні обчислення. Наприклад, популярне доповнення ScientificPython може похвалитися власними бібліотеками, які створені для обчислювальних процесів в сфері науки.

Крім розрахунків, Python дозволяє візуалізувати отримані дані, що доволі зручно.

Машинне навчання

Крім основного інструментарію, у Python є додаткові бібліотеки і фреймворки, що дозволяють працювати в області машинного навчання. Особливою популярністю користуються scikit-learn і TensorFlow. Scikit-learn відрізняється тим, що в нього вже вбудовані найпоширеніші алгоритми навчання. TensorFlow, в свою чергу – це низькорівнева бібліотека, яка відкриває можливості для створення алгоритмів користувача.

Процеси машинного навчання, засновані на мові програмування Python, допомагають реалізовувати системи розпізнавання осіб і голосу, створювати нейронні мережі, глибоке навчання і багато іншого.

1.4 Порівняння середовищ програмування мовою Python

Мова програмування Python має легкий і зрозумілий синтаксис, порівняно з іншими мовами програмування, керування пам'яттю – цілком автоматичне не потрібно хвилюватися щодо розподілу або звільнення пам'яті. Саме тому цю мову часто обирають як одну з базових для вивчення основ програмування. Все більше спеціалістів зі всього світу обирають саме цю досить універсальну мову програмування. Є багато середовищ розробки для цієї мови. Але от питання – що вибрати?

Інтегроване середовище розробки (Integrated Development Environment – IDE) – це комп’ютерна програма, призначена для розробки нового програмного забезпечення чи модифікації вже існуючого. Як випливає з назви, IDE об'єднує кілька інструментів, спеціально призначених для розробки. Ці інструменти зазвичай включають редактор, призначений для роботи з кодом (наприклад, підсвічування синтаксису і автодоповнення); інструменти збірки, виконання та налагодження; і певну форму системи управління версіями.

Більшість IDE підтримують безліч мов програмування і мають багато функцій, через що можуть бути великими, займати багато часу для завантаження і установки і вимагають глибоких знань для правильного використання.

З іншого боку, є редактори коду, які представляють собою текстовий редактор з підсвічуванням синтаксису і можливостями форматування коду. Більшість хороших редакторів коду можуть виконувати код і використовувати відлагоджувач, а кращі навіть взаємодіють із системами управління версіями.

Вимоги до середовища розробки. Отже, що нам потрібно від середовища розробки? Набір функцій різних середовищ може відрізнятися, але є набір базових речей, що спрощують програмування:

1. Збереження файлів. Якщо IDE або редактор не дають вам можливості зберегти роботу і пізніше все відкрити в тому ж стані, в якому воно було під час закриття, то не така вже це і IDE;

2. Запуск коду з середовища. Те ж саме, якщо вам потрібно вийти з середовища для запуску коду, то це не більше, ніж простий текстовий редактор;
3. Підтримка налагодження. Можливість крок за кроком виконати код є базовою функцією всіх IDE і більшості хороших редакторів коду;
4. Підсвічування синтаксису. Можливість швидко знайти ключові слова, змінні та інше робить читання і розуміння коду на порядок простіше;
5. Автоматичне форматування коду. Будь-редактор або IDE, який дійсно таким є, розпізнає «:» після while або for вираження і автоматично зробить відступ на наступному рядку.

Стає зрозуміло, що є безліч інших функцій, від яких ви б не відмовилися, але наведені вище – основні функції, якими повинна володіти добре середовище розробки.

А тепер давайте поглянемо деякі інструменти загального призначення, які можна використовувати для розробки на Python. Редактори і IDE з підтримкою Python

Eclipse + PyDev.

Тип: IDE

Сайт: www.eclipse.org

Якщо ви близькі з open-source спільнотою, то ви напевно чули про Eclipse. Він доступний для Linux, Windows і OS X, Eclipse, де є open-source IDE для розробки на Java. Існує безліч розширень, які роблять Eclipse корисним для різного роду завдань. Одним з таких розширень є PyDev (рис.1), що надає інтерактивну консоль Python і можливості для налагодження і автодоповнення коду.

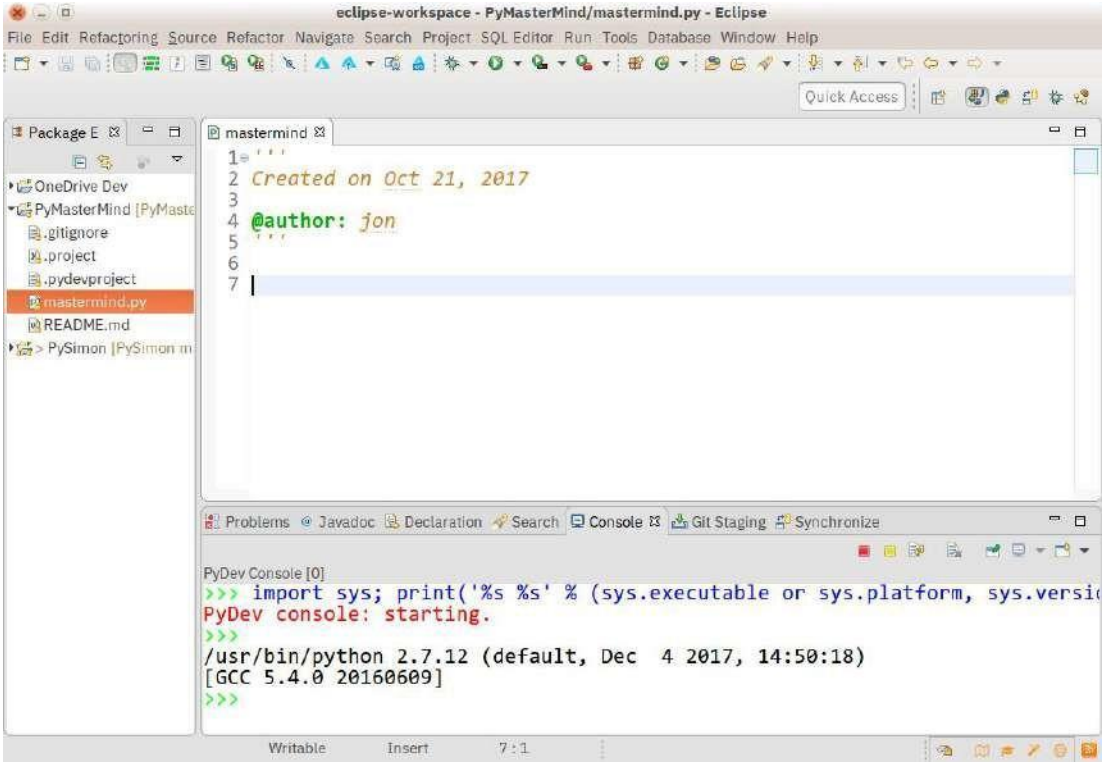


Рис. 1.1

Переваги: якщо у вас вже був встановлений Eclipse, то установка PyDev пройде швидко і гладко. У досвідченого користувача Eclipse не виникне проблем з вивченням цього розширення.

Недоліки: якщо ви тільки починаєте вивчати Python або розробку в цілому, Eclipse може стати непосильним тягарем. Пам'ятайте, ми говорили, що IDE великі і вимагають більше досвіду для повноцінного використання? Все це можна сказати про Eclipse.

Sublime Text

Тип: редактор коду

Сайт: http://www.sublimetext.com

Sublime Text (рис.2), написаний інженером з Google з мрією про кращий текстовий редактор, є вельми популярним редактором коду. Доступний на всіх платформах. Sublime Text має вбудовану підтримку редагування Python-коду, а також багатий набір розширень, званих пакетами, які розширюють можливості синтаксису і редагування.

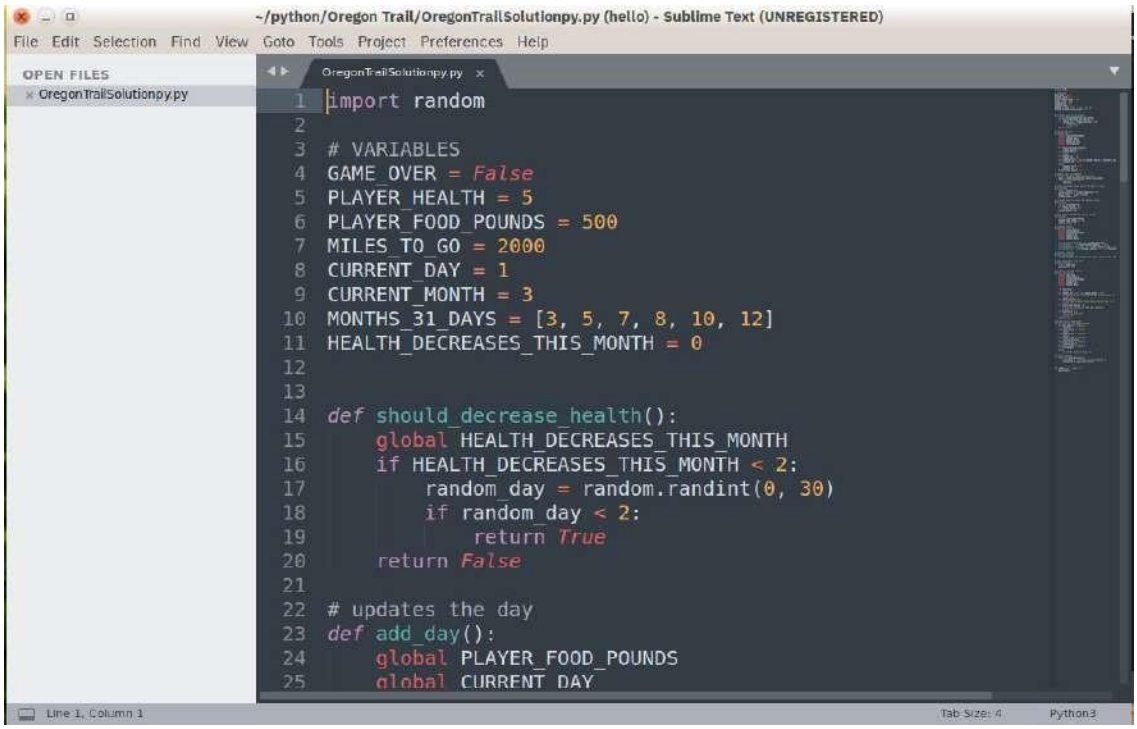


Рис. 2

Переваги: у Sublime Text велику кількість шанувальників. Як редактор коду, Sublime Text швидкий, легкий і має хорошу підтримку.

Недоліки: Sublime Text не є безкоштовним, хоча ви можете використовувати пробний період скільки завгодно. Установка розширень може перетворитися в той ще квест. Крім того, в редакторі немає підтримки налагодження і запуску коду.

Atom

Тип: редактор коду

Сайт: https://atom.io/

Доступний на всіх платформах Atom називають «хакабельним текстовим редактором 21 століття». Atom написаний з використанням Electron – фреймворка для створення кроссплатформенних додатків для робочого столу засобами JavaScript, HTML і CSS, і має безліч розширень. Підтримку Python можна також можна підключити за допомогою розширення, яке можна встановити прямо в Atom.

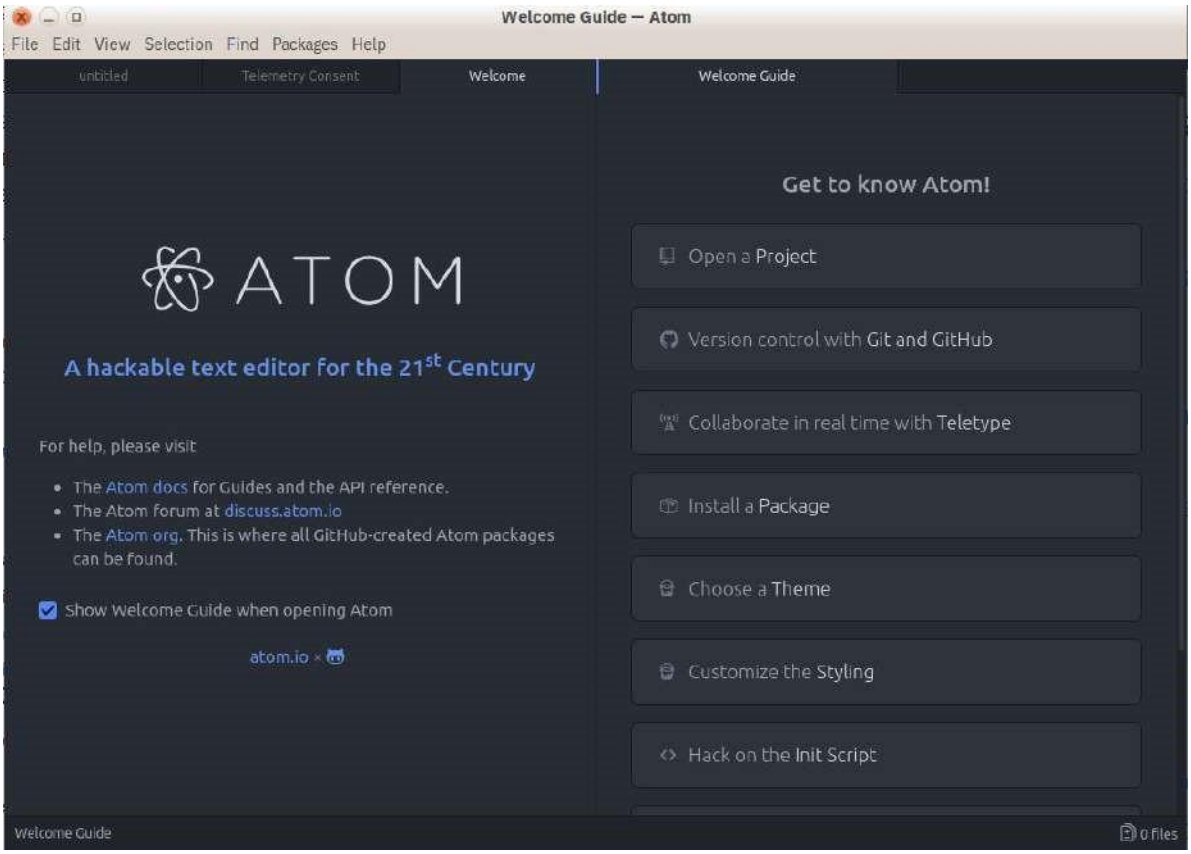


Рис. 3

Переваги: підтримка на всіх платформах завдяки Electron. Atom легкий і швидко скачується і завантажується.

Недоліки: підтримка збірки і налагодження невбудованої, а додається за допомогою розширень. Оскільки Atom написаний за допомогою Electron, він завжди працює як JavaScript-процес, а не як природний додаток.

GNU Emacs

Тип: редактор коду

Сайт: https://www.gnu.org/software/emacs/

Задовго до війни iPhone з Android, до війни Linux з Windows, навіть до війни PC з операційною системою Mac була війна редакторів з GNU Emacs (рис. 4) в якості одного з учасників військових дій. Описуваний як «розширюваний, що настроюється, самодокументований текстовий редактор», GNU Emacs існує майже так само довго, скільки і UNIX, і встиг завоювати чимало шанувальників.

Доступний безкоштовно на кожній платформі (в тій чи іншій формі) GNU Emacs використовує мову Lisp для кастомізації. Само собою, для Python теж знайдуться скрипти кастомізації.

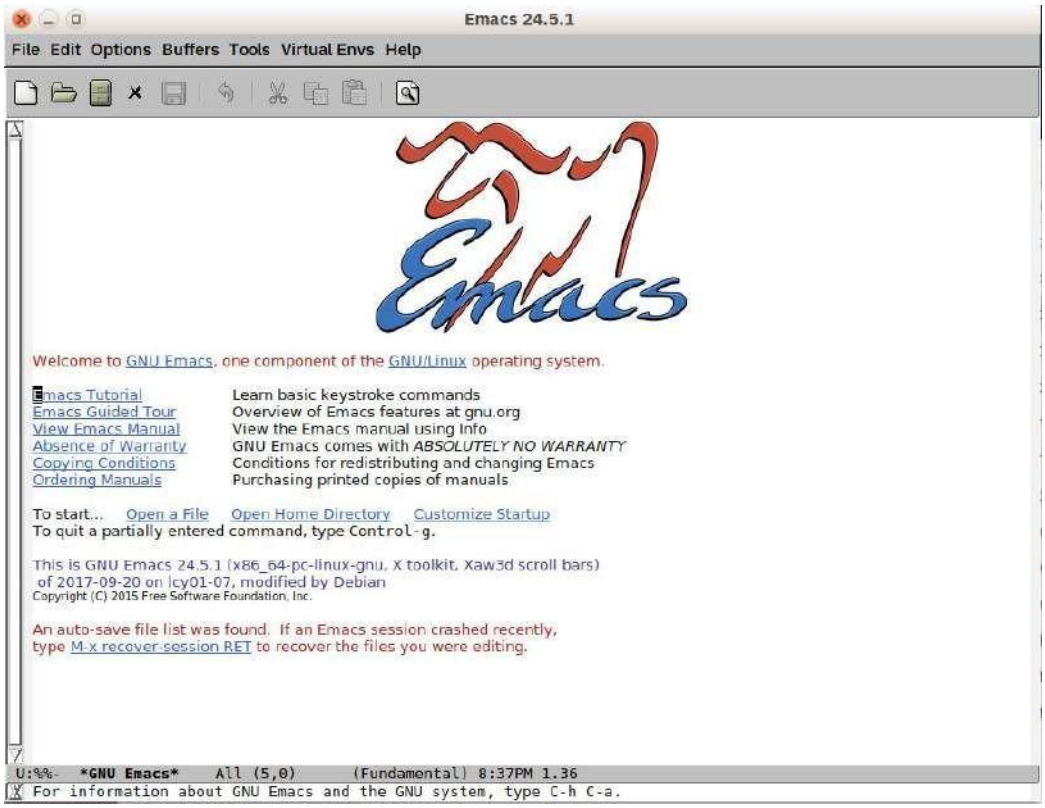


Рис. 4

Переваги: ви знайомі з Emacs, ви використовуєте Emacs, ви любите Emacs. Lisp – ваша друга мова, і ви знаєте, що з ним ви здатні на все.

Недоліки: кастомізація передбачає написання (або копіпаст) Lisp-коду в різні скрипти. Якщо таких немає, то вам, можливо, доведеться вивчити Lisp, щоб з усім розібратися.

Vi / Vim

Тип: редактор коду

Сайт: https://www.vim.org/

По інший бік барикад у війні редакторів знаходиться VI / VIM. Доступний за замовчуванням на майже кожної UNIX-системі і Mac OS X, VI завоював не меншу кількість шанувальників. VI і VIM – модальні редактори, які відокремлюють перегляд файлу від його редагування. VIM включає в себе

все, що є в VI, плюс деякі удосконалення зразок доступності розширень. Для різного роду Python-завдань можна скористатися VIMScripts.

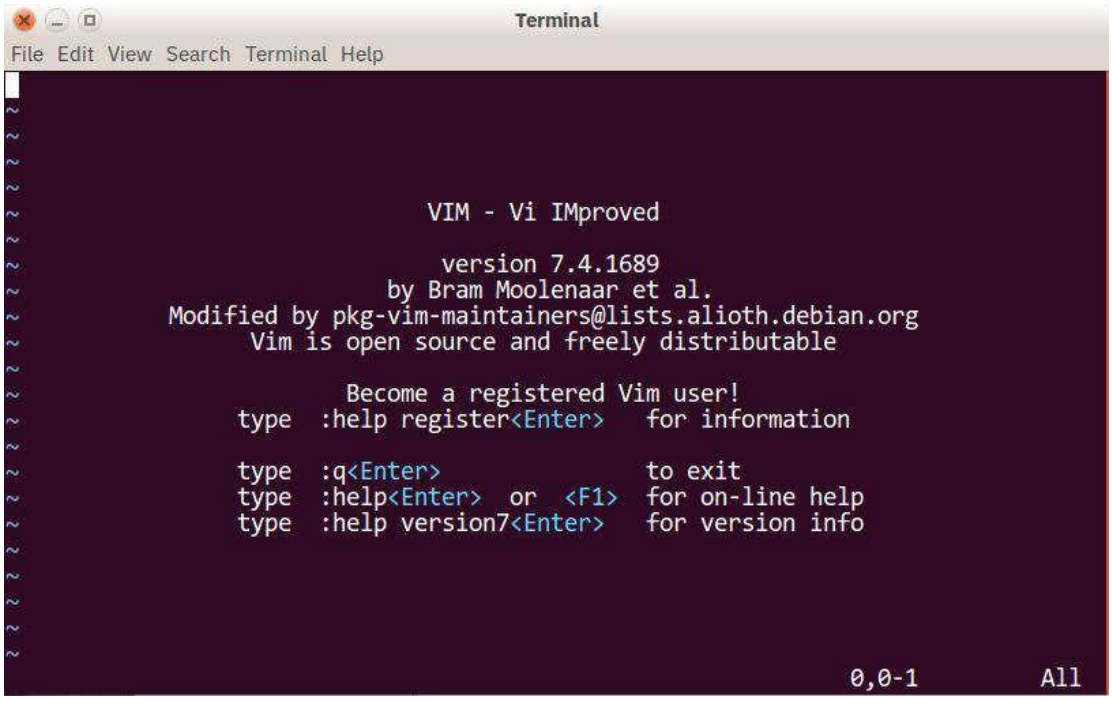


Рис. 5

Переваги: ви знайомі з VI, ви використовуєте VI, ви любите VI. VIMScripts вас не лякають, і ви знаєте, як підпорядкувати їх своїй волі.

Недоліки: як і у випадку з Emacs, вам не дуже зручно шукати або писати скрипти для додавання можливості розробки під Python, і ви не маєте ні найменшого поняття, як взагалі повинен працювати модальний редактор.

PyCharm

Тип: IDE

Сайт: https://www.jetbrains.com/pycharm/

Однією з кращих повнофункціональних IDE, призначених саме для Python, є PyCharm (рис. 6). Існує як безкоштовний open-source (Community), так і платний (Professional) варіанти IDE. PyCharm доступний на Windows, Mac OS X і Linux.

PyCharm «з коробки» підтримує розробку на Python безпосередньо – відкрийте новий файл і починайте писати код. Ви можете запускати і

налагоджувати код прямо з PyCharm. Крім того, в IDE є підтримка проектів і системи управління версіями.

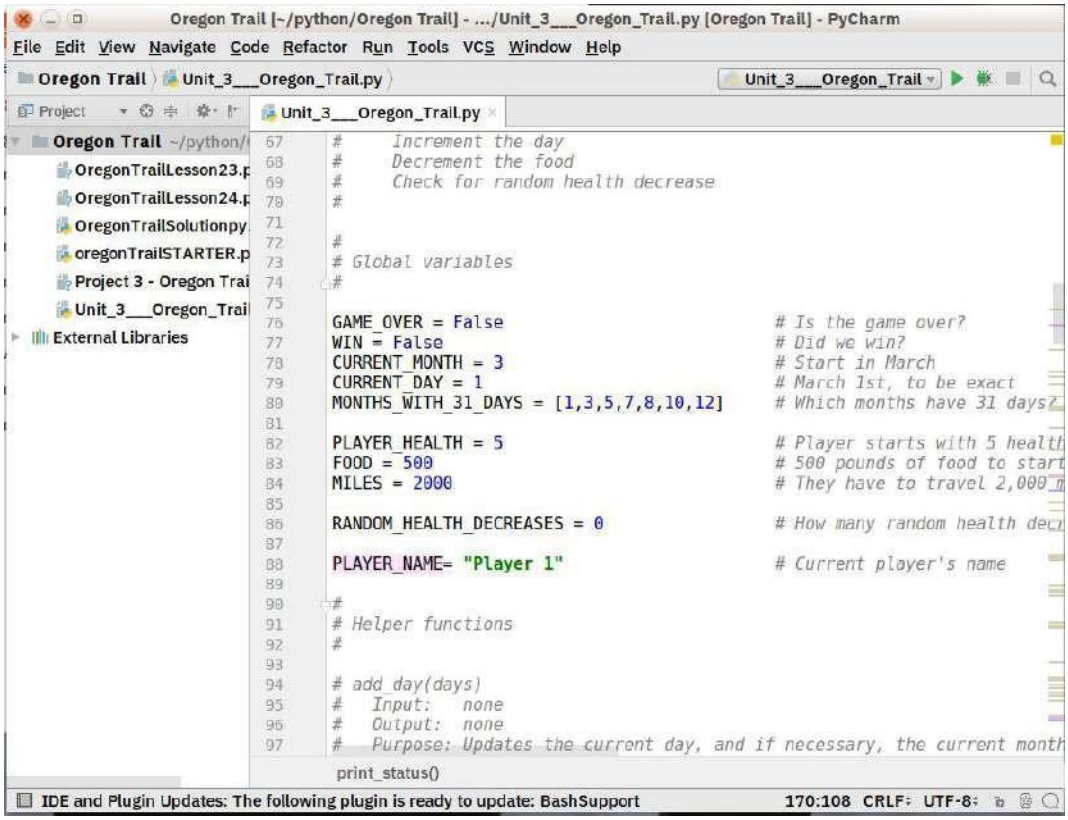


Рис. 1.6

Переваги: це середовище розробки для Python з підтримкою всього і вся, і хорошим ком'юніті. У ній «з коробки» можна редагувати, запускати і налагоджувати Python-код.

Недоліки: PyCharm може повільно завантажуватися, а налаштування за замовчуванням, можливо, доведеться підкоригувати для існуючих проектів.

Що з цього вибрати вирішувати тільки вам, але ось пара рекомендацій:

- Початківцям пітоністам слід взяти щось з найменшими можливостями кастомізації. Чим менше перешкод, тим краще;
- Якщо ви вже користуєтеся яким-небудь редактором для інших завдань, то подивіться в сторону редакторів коду;
- Ну а якщо у вас вже стоїть IDE для іншої мови, то спробуйте додати в неї підтримку Python.

Висновки до розділу 1

Мова програмування – це система позначень, що служить для точного опису програм або алгоритмів для ЕОМ. Мови програмування є штучними мовами. Від природних мов вони відрізняються обмеженим числом «слів» і дуже строгими правилами запису команд (операторів). Тому при застосуванні їх за призначенням вони не допускають вільного тлумачення виразів, характерного для природної мови.

Виділяють такі типи мов програмування як машинні, машинно-орієнтовані (мови асамблери) та машинно-незалежні (мови високого рівня). У свою чергу мови високого рівня поділяються на процедурні, логічні, об'єктно-орієнтовані.

Мова програмування Python – це універсальна сучасна МП високого рівня, до переваг якої відносять високу продуктивність програмних рішень і структурований код, що легко читається. Мова відрізняється максимально полегшеним синтаксисом. Це дозволяє вивчити її за порівняно короткий час. Ядро має дуже зручну структуру, а широкий перелік вбудованих бібліотек дозволяє застосовувати значний набір корисних функцій і можливостей. МП може використовуватися для написання прикладних програм, а також розробки WEB-сервісів.

До основник структур даних МП Python відносять списки (list), кортежі (turple), словники (dictionary), множини (set), послідовності.

Мова широко використовується в таких сферах як машинне навчання, складні обчислювальні процеси, системне програмування, бази даних, графічний інтерфейс, web розробка.

РОЗДІЛ 2. МЕТОДИЧНІ ОСОБЛИВОСТІ ВИВЧЕННЯ ПРОГРАМУВАННЯ МОВОЮ PYTHON МАЙБУТНІМИ УЧИТЕЛЯМИ МАТЕМАТИКИ

2.1. Роль і місце програмування в курсі інформатики

Одним з найважливіших напрямків підготовки будь-якого ІТ-фахівця є навчання програмуванню. Це один з фундаментальних компонентів інформатики, важливість якого для сучасного технологічного суспільства не підлягає сумніву. Елементи програмування входять навіть в освітні стандарти гуманітарних спеціальностей, таких, наприклад, як психологія і соціологія. Це говорить про те, що розробники стандартів вищої професійної освіти не сумніваються в необхідності вивчення програмування гуманітаріями, хоча б на початковому рівні. У той же час відношення до інформатики взагалі, і зокрема до програмування, з боку системи загальної освіти не настільки однозначне.

З одного боку постає відоме гасло академіка А. П. Єршова «Програмування – друга грамотність» (1981 р). У своїй статті А. П. Єршов викладає своє бачення ролі комп'ютера в навчальному процесі і показує, що «грамотність і програмування не тільки вибудовуються в паралель, з'єднуючись містками аналогій, але і доповнюють один одного, формуючи нове уявлення про гармонію людського розуму».

А. П. Єршов дивним чином передбачає ряд положень сучасного компетентного підходу, заявляючи, що крім звичайного (книжкового) знання людина повинна актуалізувати накопичені за тисячоліття операційні знання. В цьому і полягає найважливіше призначення програмування або, якщо сказати ширше, алгоритміки.

З іншого боку, існує не цілком централізована опозиція, яка протиставляє алгоритмізацію – гуманізації. В достатній мірі цей підхід виражений в структурі і змісті нових федеральних державних освітніх стандартів загальної освіти другого покоління, в яких представлений в основному технологічний компонент інформатики. Що ще більш несподівано, інформатика часто вивчається розділеною навпіл: на технологічний компонент, котрий вивчається

в рамках дисципліни «Технологія», і теоретичний компонент, перенесений в уроки математики.

Ми дотримуємося думки, що інформатика не повинна ні зводитися до технології, ні викладатися як один з розділів математики. Це означає, зокрема, що крім інформаційних технологій, елементів інформаційного моделювання і математичної основи теорії інформації повинно вивчатись і саме програмування. При такому підході виникає ряд питань: коли починати, чому вчити і що ми взагалі хочемо отримати в результаті.

Крім того, відзначимо, що нам близькі конструктивістські ідеї Ж. Піаже, С. Пейперта, А. Кея і інших дослідників, які показали, що будь-яка діяльність може знайти сенс і інтерес, якщо замінити вивчення об’єкта його конструюванням. Такі дії вимагають від учнів не тільки наявності певних знань про об’єкт, але і деяких мета предметних, операційних умінь, пов’язаних з конструюванням. Ж. Піаже відмічає, що будь-яке знання пов’язано з дією.

Як ми знаємо, підхід С. Пейперта виявився цілком конкретним: він створив мову програмування, яку зможе опанувати навіть маленька дитина. І все тому, що дії, котрі складають алгоритм, дитина, в принципі, може виконати самостійно, без жодного комп’ютера. Дуже важливо, що і результативність такої роботи програми також має цілком ясний і наочний сенс, без зайвих абстракцій: в результаті виходить графічне зображення об’єкта.

На думку С. Пейперта, програмування – це «об'єкт, за допомогою якого думають». Алан Кей, один з творців LOGO, додає, що засоби, які суттєво змінюють способи мислення, повинні бути доступні якомога раніше. А. П. Єршов, який з цікавістю спостерігав за експериментами Пейперта, зауважив: *«Хотілося б підкреслити, що мова йде не про те, щоб нав’язати учням нові, невластиві їм навички і знання, а про те, щоб проявити і сформулювати ті сторони мислення і поведінки, які реально існують, але формуються стихійно, та неусвідомлено»*.

Таким чином, стає зрозумілим, що програмування – це не самоціль. В процесі вивчення програмування (і алгоритміки взагалі) формуються

специфічні «функціональні мозкові органи» (А. Леонтьев). І дуже важливо, що ці «органи» формуються в спілкуванні і предметній діяльності. Програмування – досить своєрідна діяльність, тому і сформовані нею «мозкові органи» специфічні.

С. Пейперт вважав, що розроблений ним LOGO є моделлю мови, котра реалізує конструктивістський підхід, і згодом з'явиться безліч більш сучасних і більш продуманих мовних концепцій.

Сьогодні існує величезна кількість цікавих програмних продуктів, котрі в тій чи іншій мірі реалізують конструктивістські ідеї. Відзначимо серед них Scratch [2]; одна з сучасних реалізацій Smalltalk – Squeak [3]; тимчасові клони LOGO – Star Logo [4] і NetLogo [5]; 3D-програмування Alice [6]; фреймворк, що дозволяє створювати програми на Java в стилі, що нагадує Scratch, – Greenfoot [7]. Окремо можна виділити продукти, орієнтовані на створення ігор: засновані на Python – Pygame [8], Game Maker [9], Little Wizard [10] та інші. Всі ці проекти так чи інакше передбачають вивчення програмування, але базуються при цьому на нетрадиційному підході. Пояснимо сказане і розглянемо кілька відомих способів організації матеріалу з програмування.

Існує два різних жанри: опис мови і опис алгоритмів. Зрозуміло, неможливо говорити про мову і не сказати ні слова про алгоритми. Та й для пояснення алгоритміки не можна обійтися без мови програмування, хоча часом вибрана мова може бути дуже екзотичною, як, наприклад, в праці Д. Кнута «Мистецтво програмування». Розглянемо типову структуру книг з мов програмування.

Починаючи з найперших праць із програмування були вироблені незмінні до цього дня канони викладання. Якщо ми говоримо про імперативне програмування, то спочатку розглядаються алфавіт, прості типи, основні вбудовані функції і синтаксичні конструкції (оператори), потім структурні типи та засоби роботи з ними. Паралельно можуть вивчатися і основні алгоритми. Наприклад, у виданні 1980 року два розділи (другий і третій) присвячені вивченню Алгола-60. У першому з них йде опис синтаксису мови, а в

наступному розглядаються структури даних. Схожий спосіб викладання можна виявити в наступних книгах вивчення програмування. Всі ці книги слідують з інтервалом приблизно в 10 років між собою, але мають вищеописану канонічну структуру.

Не сильно змінюється структура побудови і в сучасних підручниках, що виходять через 30 років після першої книги з програмування. Хіба що додатково відбувається орієнтація учня на роботу з тим або іншим (зазвичай об’єктно орієнтованим) фрейм ворком.

Такий підхід виправдовує себе, якщо поставлена мета навчити фахівця, та й то лише в разі спочатку добре підготовленого і професійно орієнтованого студента.

В якості альтернативи стандартної схеми вивчення програмування ми пропонуємо скористатися конструктивістськими ідеями і відмовитися від вивчення інструменту до виникнення необхідності в ньому. Це можливо, якщо не «вивчати мову програмування», а просто програмувати. Саме так може будуватися курс програмування мовою Python.

Основне наше завдання – навчити програмувати. Ось чому можна побачити в проведених заняттях і творчі майстерні, і тематичні презентації, і створення динамічних коміксів, і елементи дослідницької діяльності.

Таким чином, хоча програмування посідає важливе місце в навчанні інформатики, ми вважаємо, що сучасні навчальні програми та плани охоплюють значно більше теорії, ніж практики. А як відомо, дисципліна вважається засвоєною тоді, коли учень чи студент вміє переносити теорію на практику. Окрім того, програмування – це швидше практична дисципліна, яка потребує педагогічних та практичних дій збоку того, хто її вивчає.

Виходячи з цього, ми вважаємо доцільним провести аналіз наявних робочих програм.

2.2. Аналіз робочих програм з інформатики для студентів спеціальностей 014 Середня освіта (Математика) та 111 Математика

Ми вже зазначали, що вся документація навчання програмуванню, а саме література, підручники, книги, навчально-виробничі та освітні програми, методичні комплекси, має схожу структуру та базується на одних і тих же ключових засадах.

Нами було проаналізовано робочі програми для студентів за спеціальностями 014 Середня освіта (Математика) та 111 Математика різних вузів України.

Виходячи з цього представимо детальний аналіз та дійдемо до конкретних висновків.

Спеціальність 111 Математика

Відповідно до діючої робочої програми для студентів факультету математики, фізики, комп'ютерних наук і технологій Вінницького державного педагогічного університету імені Михайла Коцюбинського, які навчаються за спеціальністю 111 Математика, на розділи «Основи алгоритмізації» та «Основи програмування» відводиться 134 години, з них 18 годин лекцій, 16 годин практичних і 38 годин лабораторних занять. Зокрема, на розділ основи програмування відводиться 120 годин, з них 14 годин лекцій, 14 годин практичних і 38 годин лабораторних занять.

До переліку тем, що включаються в розділи «Основи алгоритмізації» та «Основи програмування», входять наступні:

Тема 1. Етапи розв'язання задач за допомогою ЕОМ. Комп'ютерне моделювання. Моделі та їх типи. Комп'ютерна модель. Етапи побудови комп'ютерної моделі.

Тема 2. Алгоритми

Алгоритми та його основні властивості. Способи запису алгоритмів. Їх класифікація. Величини і типи величин. Операції, операнди та оператори. Базові структури алгоритмів.

Теми, що включають розділ алгоритмізації наступні:

Тема 1. Мови програмування

Поняття програми. Інструментальна мова та інтегроване середовище (система) програмування. Класифікація мов програмування. Типи програмного забезпечення. Інтерфейс та система меню середовища програмування.

Тема 2. Мова програмування Python

Особливості та призначення. Алфавіт. Основні поняття мови: вирази, операнди, операції та оператори, ідентифікатори, константи, рядки, описи. Класифікації операцій. Пріоритет операцій. Оператори. Оператор присвоювання. Оператор введення та виведення. Логічна структура програми.

Тема 3. Прості типи даних

Класифікація типів даних. Ординальні та стандартні типи даних. Діапазони значень стандартних типів. Перераховний та діапазонний тип. Стандартні функції та операції опрацювання простих типів. Процедури та функції керування екраном в текстовому режимі.

Тема 4. Організація циклів і розгалужень

Складений оператор. Оператори розгалуження. Оператор варіанту. Організація циклів. Оператори циклу: з передумовою, післяумовою, параметром. Організація завчасного виходу з циклу.

Тема 5. Структуровані та рядкові типи даних

Масиви. Одновимірні та багатовимірні масиви. Операції над масивами. Впорядкування (сортування) та пошук в масивах. Рядкові змінні. Стандартні функції для роботи з рядковими типами. Записи та множини.

Спеціальність 014 Середня освіта (Математика)

Відповідно до діючої робочої програми для студентів факультету математики, фізики, комп'ютерних наук і технологій Вінницького державного педагогічного університету імені Михайла Коцюбинського, які навчаються за

спеціальністю 014 Середня освіта (Математика), на розділи «Основи алгоритмізації» та «Основи програмування» відводиться 150 години, з них 14 годин лекцій, 24 години практичних і 42 години лабораторних занять. Зокрема, на розділ основи програмування відводиться 120 годин, з них 14 годин лекцій, 14 годин практичних і 38 годин лабораторних занять. Теми для вивчення пропонуються аналогічні до тих, які вивчаються на спеціальності 111 Математика. Зауважимо, що кількість годин на вивчення основ програмування на цій спеціальності є дещо меншою від відповідних годин на спеціальності 014 Середня освіта (Математика). Однак у програму з інформатики для спеціальності 111 Математика, на відміну від спеціальності 014 Середня освіта (Математика), входить розділ «Основи об'єктно-орієнтованого і візуального програмування», на який відводиться 114 годин, з яких 24 години лекцій, 12 годин практичних і 44 години лабораторних занять. Для студентів спеціальності 014 Середня освіта (Математика) об'єктно-орієнтоване програмування є вибірковою дисципліною за додатковою предметною спеціалізацією «Інформатика».

В цілому, основи алгоритмізації і програмування в зазначених спеціальностях вивчаються досить розгорнуто, встановлена оптимальна та доцільна кількість годин для кожної теми та виду заняття.

2.3. Організація навчальної діяльності студентів під час вивчення програмування мовою Python

Одними з основних розділів сучасних курсів інформатики, які викладаються в більшості вузів, є розділи, пов'язані з навчанням програмування, формуванням алгоритмічного мислення, підготовці до оперування з найважливішими інструментальними системами і засобами. Разом з тим в раніше опублікованих роботах неодноразово наголошувалося на необхідності вдосконалення методичних систем навчання програмуванню в зв'язку з потребою підготовки фахівців, які володіють процедурними, об'єктно-орієнтованими, логічними і функціональними підходами до розробки алгоритмів і програмування. Докладні підходи в інформатиці прийнято називати парадигмами програмування. Таким чином, стають актуальними

питання вивчення існуючих підходів до організації навчання програмуванню в вузі і розвитку курсу інформатики з метою підготовки фахівців, які володіють усіма парадигмами програмування.

Якщо говорити більш точно, для підготовки студентів вузу в галузі інформатики необхідна система курсів, заснована на інтеграції парадигм програмування, яка будується відповідно до поняття інформатики як наукової дисципліни.

Зіставляючи визначення предмета інформатики і поняття програмування, можна зробити висновок, що програмування займає одну з найважливіших частин інформатики. Тому при підготовці фахівця в цій галузі програмування повинна бути відведена адекватна частина його вивченню. У програмуванні концентруються інженерні питання реалізації алгоритму при заданих просторово-часових обмеженнях, засобами конкретної мови програмування з урахуванням всього життєвого циклу програмного продукту.

Сучасний курс інформатики повинен дати знання, які будуть базою для розуміння можливостей і обмежень використання персональних комп'ютерів і програмного забезпечення в життя суспільства. Вивчення курсу передбачає отримання фундаментальних знань в галузі інформатики. Введення декількох мов, а, тим більше, парадигм програмування дозволяє адаптувати отримані знання до мінливого стану в сфері нових інформаційних технологій, що, в свою чергу, дозволяє на новому якісному рівні використовувати інформаційні технології в навчальному процесі, надає можливість реалізувати необхідну модель підготовки студентів.

Зміст інформаційної підготовки студентів відбивається в двох її структурних складових: компоненті освіти і компоненті навчання. Причому компонента освіти призначена для формування загальних знань про основні принципи інформатики та узагальнених способах побудови, функціонування та використання інформаційних технологій. Компонента освіти становить теоретичну частину змісту інформаційної підготовки. Компонента навчання повинна формувати вміння і навички роботи в конкретних умовах застосування

сучасних інформаційних технологій. Така компонента містить практичну частину інформаційної підготовки.

Курс програмування на основі вивчення певної методології розробки алгоритмів відповідає, з одного боку, вимогам, закладеним як в компоненті освіти, так і в компоненті навчання. З іншого боку, він покликаний дати необхідні знання про мову програмування, яка лежить в основі побудови інформаційних технологій на сучасному етапі розвитку інформатики.

В даний час система курсів з інформатики розподілена на два етапи підготовки студентів (бакалаврату і магістратури). Курс програмування на основі вивчення мови однієї з парадигм програмування повністю охоплює загальноосвітню підготовку (нижній щабель бакалаврату) і другий ступінь (магістратуру) за напрямками науки.

Відбір змісту системи курсів інформатики, заснованих на інтеграції парадигм програмування, повинен здійснюватися відповідно до спеціальних методичних принципів, основні з яких перераховуються далі.

1. Наукова строгість і послідовність курсу, яка передбачає несуперечливість і логічну послідовність викладу матеріалу. Для практичної реалізації даного критерію відбору змісту визначені критерії наукової строгості і послідовності навчального матеріалу: кожна тема повинна бути викладена логічно, реалізація кожної теми повинна відповідати оцінці наукового рівня і характеристикам логічної строгості.

2. Системність наукових знань. Основні положення цього критерію зводяться до того, що кожне основне поняття повинно мати чітко визначене місце в системі понять всього розділу, виклад основних ідей і понять повинен бути зроблений з використанням достатнього набору відповідних факторів, методи, використовувані в системі курсів інформатики, повинні забезпечувати раціональне рішення практичних завдань.

3. Принцип доступності забезпечується поступовістю переходу від простого до складного, посильною і доцільною термінологією і символікою, відповідністю наявного запасу знань, умінь і навичок.

4. Принцип практичної спрямованості теоретичного матеріалу. Даний принцип полягає в тому, що повинен існувати чіткий зв'язок теоретичного матеріалу з практикою, причому не тільки в якості його використання при вирішенні навчальних завдань, а й з практикою, як видом людської діяльності. Крім того, зміст повинен забезпечувати придбання у учнів практичних навичок використання отриманих знань в області програмування і алгоритмізації.

5. Принцип відповідності цілям навчання, що спирається на те, що кожне поняття або метод, що входять в зміст навчання інформатики, повинні відповідати певним цілям, яких необхідно досягти в процесі навчання, а також бути орієнтованими на залучення учнів до програмування з використанням всіх можливих парадигм.

6. Вивчення матеріалу в єдності теорії, технології і техніки, що має на увазі використання взаємозв'язку між різними аспектами інформатики (теоретичним, технологічним і технічним), використання тріади «модель-алгоритм-програма», яка лежить в основі застосування методології інформатики в різних сферах людської діяльності.

Відзначимо, що при визначенні змісту навчання інформатики необхідно враховувати складну структуру співвідношень між знаннями уміннями і навичками, які в навчальної діяльності студента виступають в діалектичній єдності і характеризують процес формування понять. Зміст будь-якого навчального предмета – це завжди певна інформація про явища або методах діяльності, характерних для даної області.

2.3.1. Вибір методів, форм і засобів навчання під час вивчення програмування мовою Python

Істотних досліджень вимагає методика навчання інформатики в разі базування відповідної методичної системи на інтеграції різних парадигм програмування. При цьому під методом навчання в вузі розуміються

впорядковані способи взаємопов’язаної діяльності викладача і студента, спрямовані на досягнення поставлених цілей навчання конкретної наукової дисципліни.

Класифікації методів навчання відрізняються один від одного критерієм, покладеним в основу кожного з них.

Розглянемо класифікації методів навчання, котрі ми рекомендуємо використовувати для навчання основам алгоритмізації і програмування за вказаними спеціальностями.

За способом передачі інформації від викладача до студента доцільно використовувати вербальні, наочні і практичні методи навчання. При навчанні курсам і розділам програмування потрібно використовувати вербальні (при викладі лекційного матеріалу) і практичні (виконання лабораторних робіт, практикумів, рішення задач) методи, причому основний акцент робиться на практичні методи, в процесі застосування яких студенти не тільки отримують нові знання, а й набувають практичні навички.

Викладач при цьому інструктує, вказує цілі роботи, спрямовує і перевіряє хід її виконання. В діяльності студентів переважає практична робота (речові і розумові дії), в ході якої особливу роль відіграє самостійний розумовий процес, що дозволяє здійснити пошук даних і парадигми рішення задачі.

За основними видами дидактичних проблем, що вирішуються на занятті, ми рекомендуємо використовувати методи придбання знань, формування умінь, застосування знань, методи творчої діяльності та методи перевірки знань, умінь і навичок. Відзначимо, що всі перераховані методи прийнятні для використання при навчанні мови програмування Python на базі інтеграції вищезазначених парадигм.

Слід зазначити, що дуже часто методика навчальної діяльності являє собою ітераційний поступальний процес. Такі висновки дозволяють запропонувати метод, застосування якого доцільно при навчанні мови програмування Python. Йдеться про ітераційний метод навчання.

Розглядаючи ітерацію як покрокове наближення до певної мети, можна застосувати метод ітерації як при викладі лекційного матеріалу, так і в процесі виконання лабораторних робіт. Тим більше, що специфіка завдань, призначених для виконання на лабораторному практикумі, цілком відповідає поступальному ітераційному процесові, який виражається в побудові ряду алгоритмів і програм вирішення завдання, причому кожен наступний алгоритм є уточненням або розширенням попереднього. Таким чином, побудова підсумкової програми із застосуванням однієї з парадигм програмування являє собою ітераційний процес, на кожному кроці якого відбуваються деякі зміни, що і дозволяє нам застосувати ітераційний метод навчання.

Згідно із зазначеними положеннями, послідовність викладу лекційного матеріалу залежить від порядку практичних і лабораторних робіт. З огляду на їх ітераційний характер, виклад лекційного курсу також має сенс будувати на основі ітераційного методу.

У разі практичного застосування подібної методики навчання реалізується не на основі поступового вивчення нових структур і операторів однією з можливих парадигм програмування, а за допомогою поступального ітераційного процесу уточнення і розширення можливостей програмної реалізації системи, що моделюється. Причому введення нових структур даних і можливостей мови програмування обґрунтовується з точки зору їх необхідності для вирішення нового завдання.

В якості засобів навчання ми рекомендуємо використовувати всі типи засобів наочності, а саме зорові (картинки, таблиці, схеми, презентації), слухові (аудіо-записи) та комбіновані (відео записи, відео-презентації, інтерактивна дошка).

Найбільш популярним засобом сьогодні вважається інтерактивна дошка.

"Інтерактивна дошка" – це сучасний мультимедіа-засіб, який, володіючи всіма якостями традиційної шкільної дошки, має більш широкі можливості графічного коментування екранних зображень; дозволяє контролювати і проводити моніторинг роботи всіх учнів одночасно; природним чином (за

рахунок збільшення потоку пропонованої інформації) збільшити навчальне навантаження студента в класі; забезпечити ергономічність навчання; створювати нові мотиваційні передумови до навчання; вести навчання, побудоване на діалозі; навчати за інтенсивними методиками з використанням кейс-методів.

2.3.2. Навчально-методичний комплекс з основ програмування мовою Python.

В результаті роботи було розроблено навчально-методичний комплекс з розділу «Основи програмування», який вивчається в курсі інформатики майбутніми вчителями математики. В якості основної мови програмування обрано Python. Основні складові цього комплексу розміщено на розробленому нами блозі «Основи програмування мовою Python» (рис. 2.1).

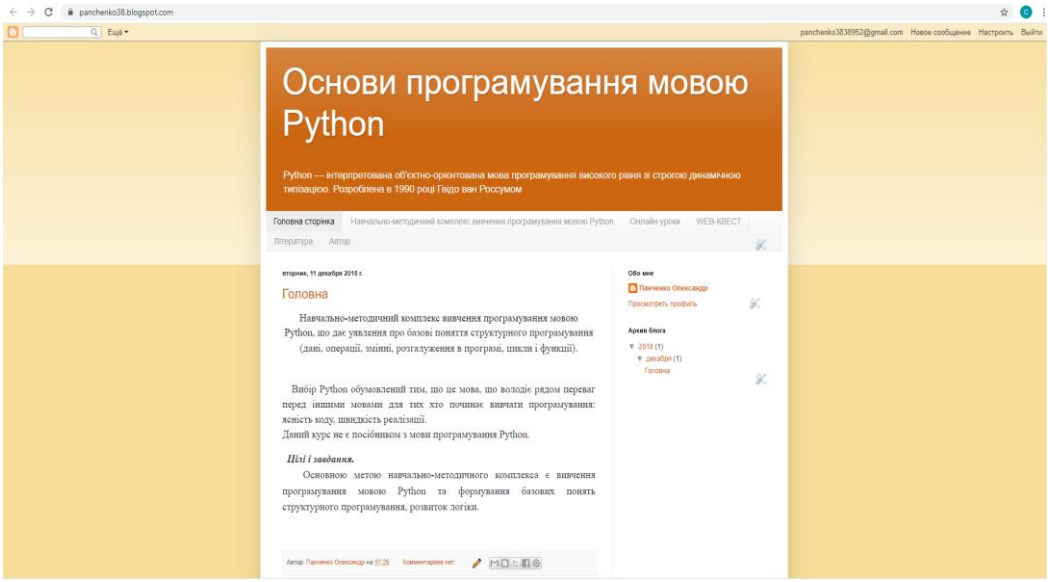


Рис. 2.1. Блог для студентів для ознайомлення з комплексом.

Було розроблено теоретичні матеріали для вивчення тем даного розділу відповідно до робочої програми.

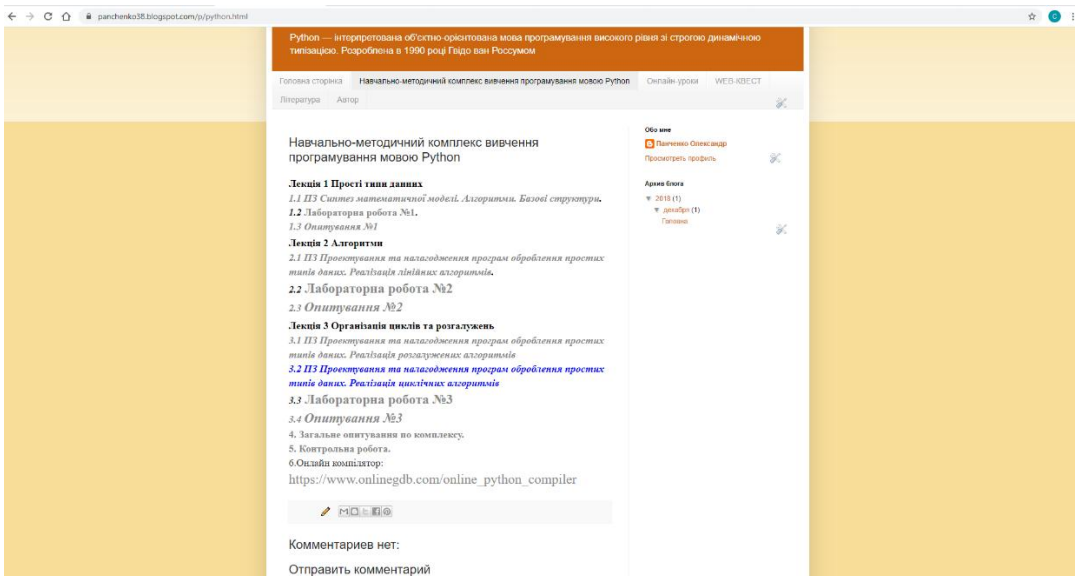


Рис. 2.2. Навчально-методичний комплекс

Студентам запропоновані відео-уроки з відповідних тем для підняття інтересу до навчання та наочності. Після курсу є адреса по якій студенти можуть перейти та виконувати практичні вправи, та пробувати виконувати приклади з лекцій.

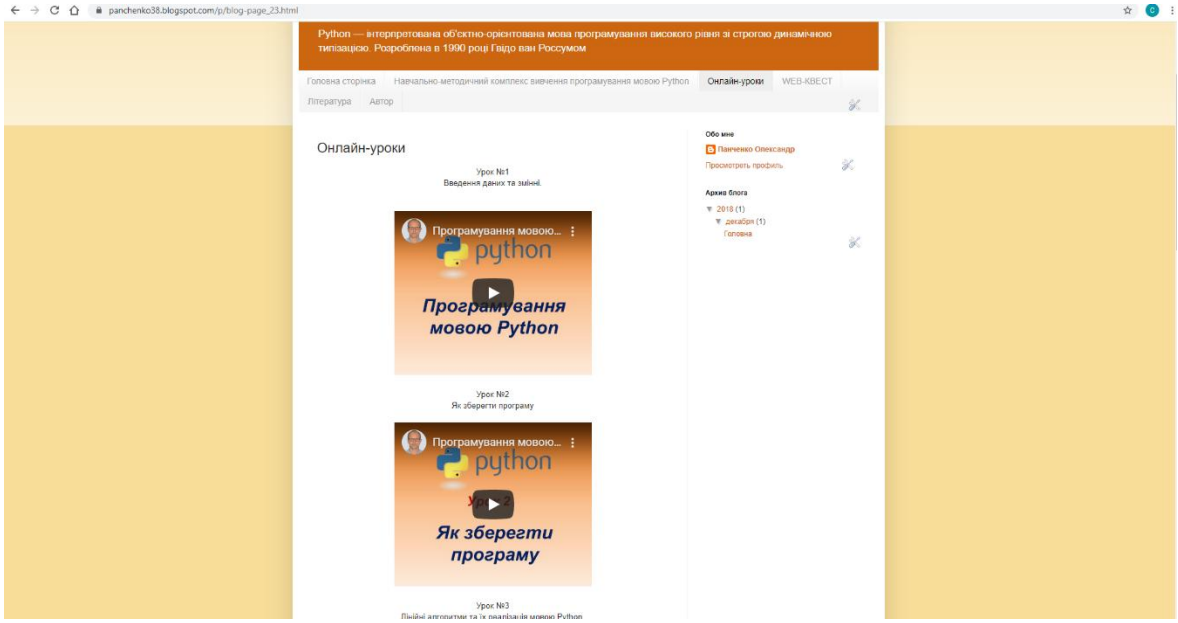


Рис. 2.3. Онлайн уроки.

Після кожної теми можна пройти онлайн тестування та дізнатися свій результат на наступному занятті в університеті.

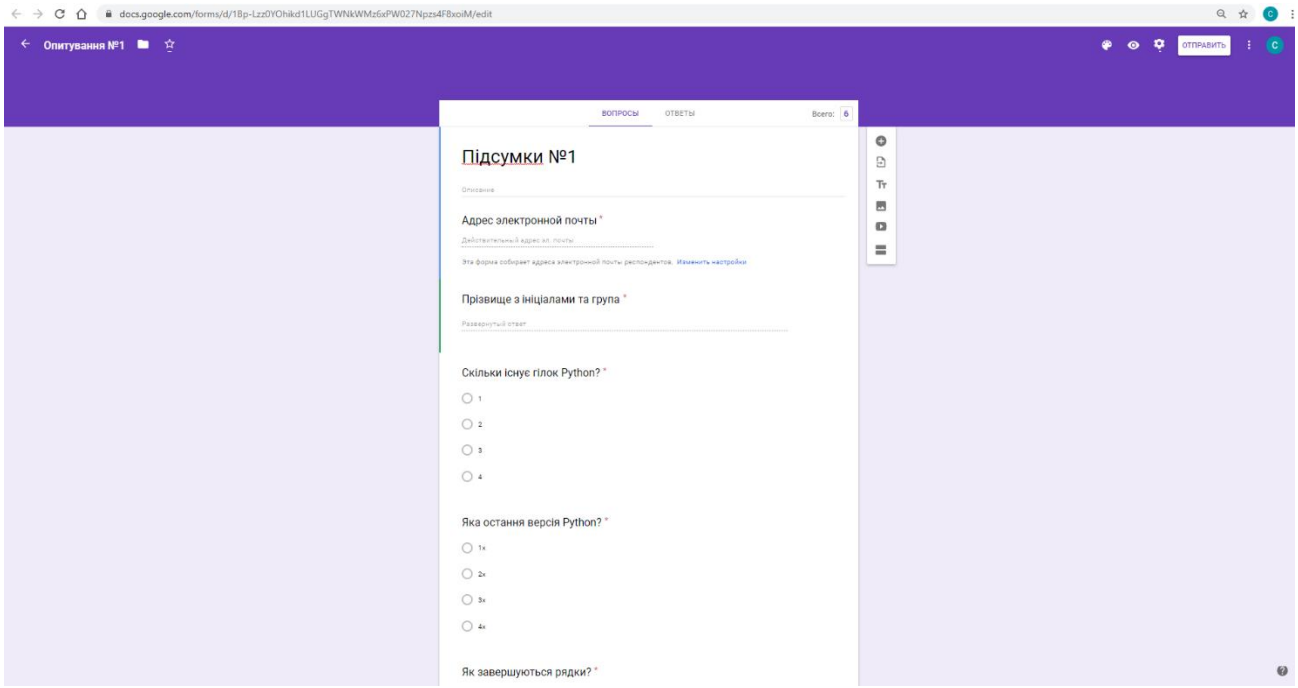


Рис. 2.4. Тестування

2.3.3. Шляхи підвищення пізнавальної мотивації навчальної діяльності студентів

В умовах розбудови національної системи вищої педагогічної освіти особливо актуальною стає професійна спрямованість навчання у педвузі та посилення зв'язку змісту навчання з повсякденним життям, що вимагає збільшення у навчальних планах долі курсів інтегративного характеру, які сприятимуть формуванню міжпредметних та міжгалузевих зв'язків, вихованню висококваліфікованих педагогів, цілеспрямована активна діяльність яких у значній мірі визначатиме майбутнє нашого суспільства. Одним з таких курсів є курс вивчення мови програмування Python, який посідає центральне місце у навчальних планах студентів (II - курсу) спеціальності *014.04 Середня освіта (Математика)*, так і за своїм значенням, втілюючи у собі практичну спрямованість курсів математики та інформатики.

Тенденція до збільшення частки самостійної навчальної та науково-педагогічної діяльності студентів із одночасним зменшенням аудиторного навантаження вимагає таких форм навчальної роботи, що активізують пізнавальну діяльність студентів. Активізація пізнавальної діяльності студентів є одним з пріоритетних напрямків досліджень педагогіки вищої школи, оскільки в ній містяться джерела багатьох проблем формування особистості майбутнього вчителя: розвиток пізнавальних інтересів, самостійності, ініціативності, цілеспрямованості, відповідальності, вольових якостей тощо.

Період навчання у закладі вищої освіти (ЗВО) – це підготовка до трудової діяльності, яка у найближчому майбутньому повинна стати для молодої людини основним джерелом існування та засобом особистісної самореалізації.

Формування світогляду, що базується на ставленні до людини як до найвищої цінності, стимулювання гармонійного розвитку й прояву творчого потенціалу особистості в праці - основа гуманізації сучасної освіти. Але вона може залишитися набором добрих намірів, якщо на практиці не вживати заходів для підвищення ефективності навчального процесу, поліпшення умов

взаємодії викладача і студента, переведення студента на рівень активного суб'єкта навчального процесу.

Не існує чітких методик або вказівок щодо підвищення ефективності навчального процесу, які були б універсальними для кожного вищого учбового закладу. Їх вибір залежить від умов, за яких він відбувається, рівня активності студентів даного ЗВО, включення студентів у науково-пізнавальну діяльність та багатьох інших факторів.

Мотивація студентів до навчання є однією з основних складових навчально-виховного процесу.

Поняття «навчальна мотивація» ми розглядаємо як процес, метод і засіб спонукання студентів до продуктивної пізнавальної діяльності, активного освоєння змісту освіти.

Через мотивацію педагогічні цілі швидше перетворюються на цілі самих студентів. Через зміст навчальної діяльності формується певне ставлення студентів до навчальної дисципліни й усвідомлюється його ціннісна значущість для особистісного, в тому числі інтелектуального розвитку.

Мотивація навчальної діяльності вимагає:

– оволодіння студентом знань в майбутній діяльності;

– створення умов для розвитку мотивів;

– зацікавленість студента до досліджуваного ним матеріалу;

– організація практичної та іншої навчальної діяльності;

– підтримання прагнення у студентів до саморозвитку та самоосвіти.

Таким чином, пріоритетними у навчальній діяльності студента стають:

– самостійна постановка мети і усвідомлення значимості виконуваної діяльності;

– самостійне освоєння базових знань і умінь;

– формування і розвиток інтелектуально-пізнавальних процесів, професійної мотивації для досягнення успіху.

У навчанні процес мотивації є безперервним, оскільки вивчення будь-якого навчального матеріалу закладає основу для викладання нового. Мотивація не повинна завершуватись при проходженні чергового відрізку навчання, вона має зберігатися і підсилюватися для подальшого навчання.

Процес мотивації у навчанні має характер певного циклу, він повторюється на кожному навчальному відрізку на вищому рівні. Саме тому, забезпечення викладачем кожного етапу мотиваційного циклу у навчанні має здійснюватись за допомогою спеціальних прийомів і методів. Серед основних методів і прийоми мотивації, а також стимулювання навчання можна виділити такі, як:

- комунікативна атака;
- доведення та переконання;
- сугестія (навіювання);
- подолання перешкод;
- делегування;
- закріплення позитивного враження.

Таким чином, мотивація повинна розглядатися як процес спрямовуючий, регулюючий і активізуючий діяльність суб’єкта навчання.

Відповідно до сучасних уявлень можна виділити кілька комунікативних методик, які можуть використовувати у навчальному процесі:

- це стимулювання когнітивних процесів (вирішення проблемних завдань, використання рольових та ділових ігор тощо);
- це використання інтерактивних форм та прийомів навчання, що беруть до уваги інтерес, мотивацію студентів до навчальної діяльності;
- це моделювання та імітація реальних чи віртуальних ситуацій;

- це взаємодія викладача у межах малої творчої групи студентів за принципом єднання індивідів із різним креативним та інтелектуальним потенціалом.

Викладач є носієм необхідної для студентів інформації під час вивчення нового матеріалу, виконує функції спостерігача, коли студенти працюють у малих творчих групах, та надає їм індивідуальну консультаційну допомогу.

Викладач є консультантом у випадках, коли студенти працюють над певною ситуаційною вправою і потребують порад та рекомендацій щодо змісту завдань.

Викладач має бути психологом і психофізіологом, визначати настрій, самопочуття й активність студентів, уміти за допомогою педагогічних і психологічних прийомів підтримувати високий рівень їх фізичної та розумової працездатності.

В сучасному світі значно зростає роль ВНЗ у навчанні студента самостійно вчитися. Самостійна робота студента, яка є суттєвим елементом навчального процесу поряд з аудиторним навчанням, набуває великого значення.

Самостійна робота студента - це навчальна діяльність студента, яка планується, виконується за завданням, під методичним керівництвом і контролем викладача, але без його прямої участі. Тільки знання, здобуті власною працею, є міцними, глибокими і дієвими. СРС формує навички самостійної діяльності взагалі, що є конче необхідним у будь-якій професійній діяльності, виробляє здатність самостійно приймати відповідальні рішення, знаходити оптимальний вихід зі складних ситуацій.

Від методів навчання значною мірою залежить розвиток студентів і якість засвоєння ними знань, і набуття навичок самостійної роботи.

Методи навчання оцінюються як один із найважливіших компонентів загальної структури навчально-виховного процесу.

Викладачам близькі та зрозумілі такі методи навчання як розповідь, бесіда, лекція, показ ілюстрацій, показ демонстраційних матеоіалів, виконання вправ, завдань тощо.

Результат діяльності викладачів і студентів не завжди такий, яким він уявляється у субєктивній формі мети. У цій взаємодії мети, засобу і результату найважливішим є аналіз пізнавальної діяльності студентів, а також керівництво нею з боку викладача. Кожен викладач повинен памятати, що методи не є незмінними.

Знаходження відповіді на питання "як підвищити ефективність навчального процесу, досягти високого інтелектуального розвитку студентів, забезпечити оволодіння ними навичок саморозвитку особистості". Значною мірою цього можна досягти використовуючи сучасні інноваційні технології, зокрема технології інтерактивного навчання, перетворюючи, таким чином, традиційний урок в інтерактивний.

Особливістю інтерактивного навчання є підготовка молодої людини до життя і громадянської активності в громадянському суспільстві і демократичній правовій державі. Це вимагає активізації навчальних можливостей студента, замість переказування абстрактної, "готової" інформації, відірваної від їх життя і суспільного досвіду.

Для зміцнення контролю над ходом процесу навчання за умов використання інтерактивної моделі навчання викладач також повинен попередньо добре підготуватися:

- глибоко вивчити і продумати матеріал, у тому числі додатковий, наприклад, - різноманітні тексти, зразки документів, приклади, ситуаційні завдання для груп тощо;
- старанно спланувати і розробити заняття: визначити хронометраж, ролі учасників, підготувати питання і можливі відповіді, виробити критерії оцінки ефективності заняття;

- мотивувати студентів до вивчення шляхом добору найбільше цікавих випадків, проблем;
- оголошення очікуваних результатів (цілей) заняття і критеріїв оцінки роботи студентів;
- передбачити різноманітні методи для привернення уваги студентів, настроювання їх на роботу, підтримання дисципліни, необхідної для нормальної роботи аудиторії.

Якщо спробувати дати визначення поняття інтерактивна технологія навчання, то - це така організація навчального процесу, за якої неможлива неучасть студента у колективному взаємодоповнюючому, заснованому на взаємодії всіх його учасників процесі навчального пізнання: або кожен студент має конкретне завдання, за яке він повинен публічно прозвітуватись, або від його діяльності залежить якість виконання поставленого перед групою завдання . Інтерактивні технології навчання включають в себе чітко спланований очікуваний результат навчання.

Така групова навчальна діяльність сприяє формуванню у студентів позитивного ставлення до навчання, розвиває вміння пристосовуватись до умов роботи в групах і забезпечує високу загальну активність студентського колективу.

На сучасному етапі розвитку суспільства обсяг та складність інформаційних потоків з кожним роком збільшується. Тому традиційна система навчання у вищих закладах освіти потребує постійного удосконалення на основі сучасних досягнень науки та техніки, що пов'язано з поліпшенням методики організації та проведення навчального процесу. Важливим напрямом підвищення ефективності навчально-пізнавального процесу є використання технічних засобів навчання , в тому числі комп'ютерної техніки .

Використання технічних засобів навчання надає навчально-методичній роботі зі студентами більш насичений, динамічний, творчий та інтенсивний характер. Сучасна освіта базується в основному на вербальному способі

передачі знань, де переважає сприймання усної інформації, яка перевантажує роботу слухового аналізатора. При цьому візуальний канал використовується мало, виникає сенсорне голодування, що значно знижає творчий характер навчальної діяльності.

В умовах стрімкого зростання інформаційних потоків і збільшення дефіциту навчального часу аудіовізуальні засоби дозволяють за один і той же термін часу викласти і засвоїти значно більший обсяг навчальних знань. При цьому якість інформації, що засвоюється студентами, підвищується за рахунок її наочності, виділення в графіках, схемах, слайдах, головних структурних елементах процесів і явищ. Використання технічних засобів навчання у викладанні навчальних дисциплін дозволяє збільшити обсяг інформації, яку необхідно запам'ятати, приблизно на 35% і підняти ефективність занять на 20%, дозволяє значно підвищити пізнавальну діяльність студентів, дає можливість надати до навчального процесу додаткову інформацію.

В даний час у всьому світі на перший план в освіті виходить застосування технологій електронного навчання.

В останні роки все частіше відзначається зниження ефективності традиційного навчання. Жорстка регламентація діяльності студентів на заняттях, примусовість навчальних процедур, часто призводить до нерозуміння ними цілей своїх дій, до відсутності усвідомлення необхідності досліджуваного матеріалу та його практичної значущості . У зв'язку з чим, у студентів спостерігається відсутність навчальної мотивації, не сформованість навичок планування своєї діяльності.

Сучасні педагогічні технології, і більшою мірою технології електронного навчання є особистісно-орієнтованими, і спрямовані на розвиток індивідуальних ресурсів студентів.

На відміну від подання знань в готовому вигляді при традиційному навчанні, технології електронного навчання передбачають підвищення рівня самостійної роботи учнів в індивідуальному темпі з одного боку, надаючи

можливості для широкого спілкування з іншими студентами та спільного планування своєї діяльності з іншого.

2.4. Експериментальна перевірка ефективності розробленої методики та аналіз результатів дослідження.

На початку нашого дослідженння (2018–2019 н.р.) нами було опитано 16 студентів 1-го курсу спеціальності 014 Середня освіта (Математика) та 7 студентів 1-го курсу спеціальності 111 Математика: «Чи хочу я вивчати мову програмування Python?».

Таблиця 2.1

Чи хочу я вивчати мову програмування Python?			
	Так	Ні	Невпевнений
014 Середня освіта (Математика)	75%	6,25%	18,75%
111 Математика	71,42 %	14,29 %	14,29 %

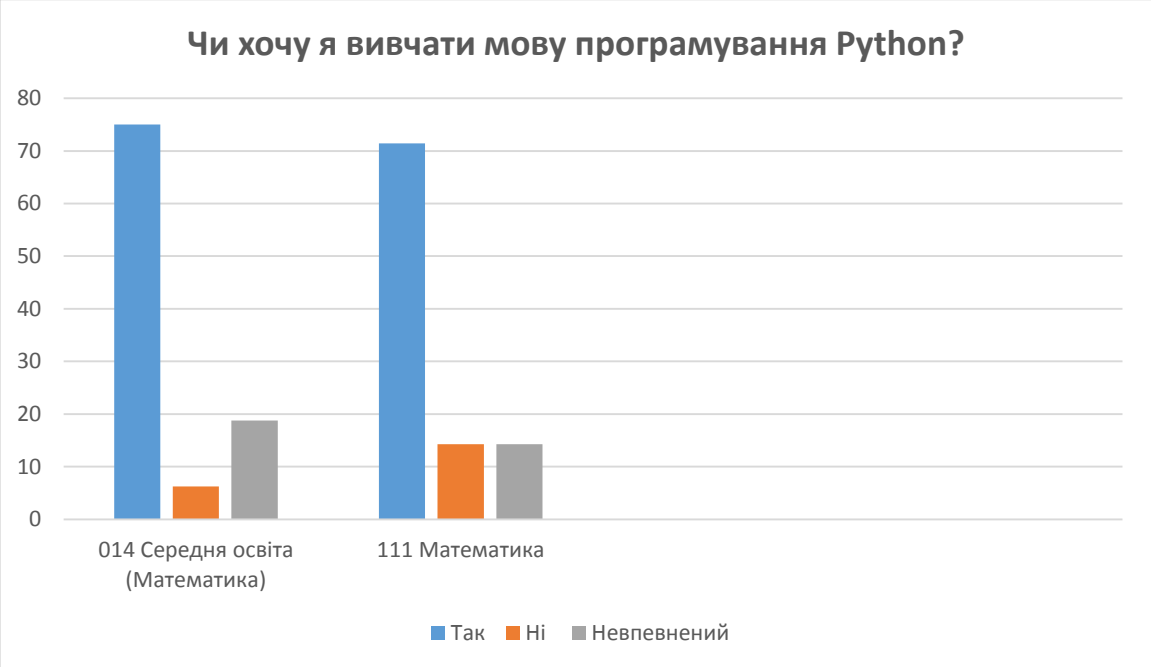


Рис. 2.5. Результати опитування студентів «Чи хочу я вивчати мову програмування Python?»

Під час проходження асистентської практики ми поставили перед собою мету експериментально перевірити ефективність розробленої нами методики. Для цього було використано розроблений нами блог із відповідним навчально-методичним комплексом, підібрані методи, форми і засоби навчання. Після цього ми провели контрольну роботу для визначення дослідницьких

компетентностей студентів 2 курсу спеціальності 014 Середня освіта (Математика) на лекційних заняттях інформатики після вивчення тем «Прості типи даних» та «Організація циклів та розгалужень». В контрольній роботі студентам було запропоновано 10 завдань. Наведемо приклади завдань:

КОНТРОЛЬНА РОБОТА

1. Які типи даних ви знаєте? Опишіть їх.
2. Чи можна перетворити дробове число на ціле? ціле на дробове? У яких випадках можна рядок перетворити на число?
3. Наведіть приклади операцій. Для чого призначена операція присвоєння?
4. Які існують правила і рекомендації для іменування змінних?
5. Напишіть програмний код, в якому у випадку, якщо значення якоїсь змінної більше 0, виводилося б спеціальне повідомлення (використовуйте функцію **print**). Один раз виконайте програму при значенні змінної більше 0, другий раз - менше 0.
6. Вдоскональте попередній код за допомогою гілки **else** так, щоб залежно від значення змінної, виводилося або 1, або -1.
7. Придумайте програму, в якій би використовувалася інструкція **if-elif-else**. Кількість гілок повинна бути як мінімум чотири.
8. Створіть скрипт (і збережіть як файл **data.py**), який би запитував у користувача:

- Його ім'я: "Як тебе звати?"
- Вік: "Скільки тобі років?"
- Місце проживання: "Де ти живеш?"

А потім має вивести три рядки

- "Це *ім'я*"
- " Це *вік*"
- "Він живе в *місце проживання*"

(замість слів *ім'я, вік, місце проживання* повинні бути відповідні дані, що введені користувачем).

9. Напишіть програму за наступним описом:

1. двом змінним присвоюються числові значення;
2. якщо значення першої змінної більше другої, то знайти різницю значень змінних (відняти від першої другу), результат присвоїти третій змінній;
3. якщо перша змінна має менше значення, ніж друга, то третю змінну пов'язати з результатом суми значень двох перших змінних;
4. у всіх інших випадках, присвоїти третій змінній значення першої змінної;
5. вивести значення третьої змінної на екран.

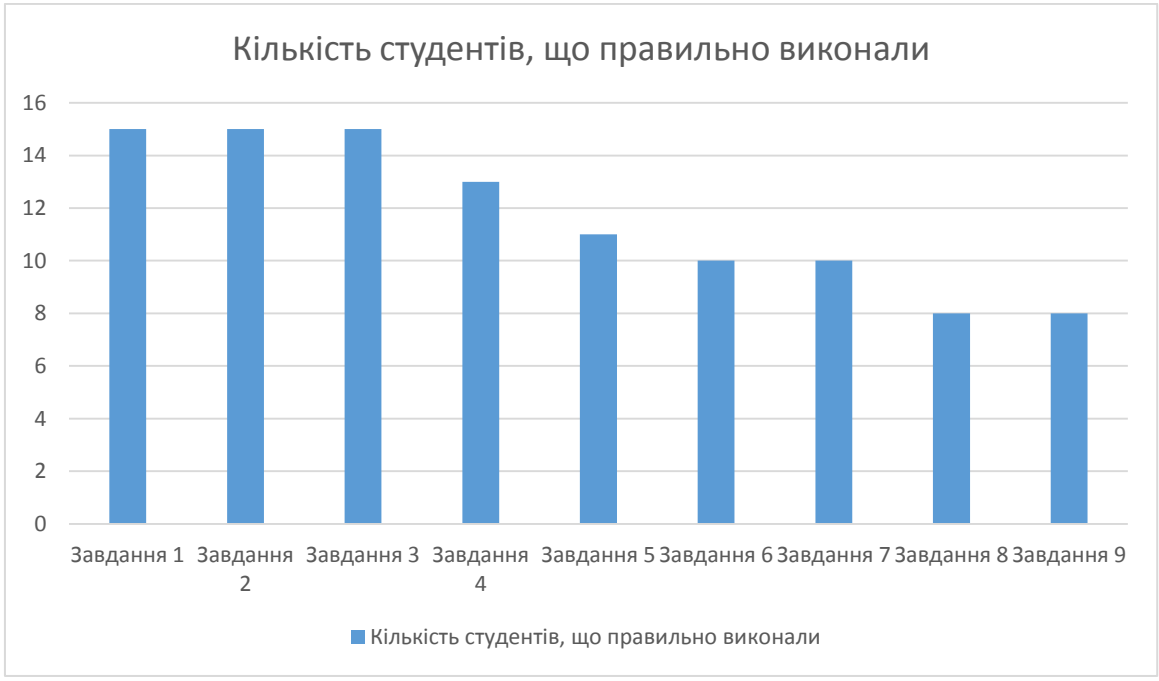


Рис. 2.6. Результати виконання контрольної роботи студентами.

Результати виконання завдань студентами наведені на рис. 2.6. Відповідно до них ми можемо зробити висновок, що після проходження комплексу 16-ма студентами групи 2 СОМ, більшість студентів засвоїла матеріал дуже добре. Перших три завдання правильно виконали всі студенти, завдання середнього рівня виконала більшість студентів і завдання високого рівня складності виконала половина студентів групи.

Також ми провели тестування серед студентів групи окремо з кожної теми. Студенти показали відмінні та дуже добрі результати. Після

проходження тестування з теми «Прості типи даних» 10 студентів отримали оцінку 5, та 6 студентів оцінку 4.

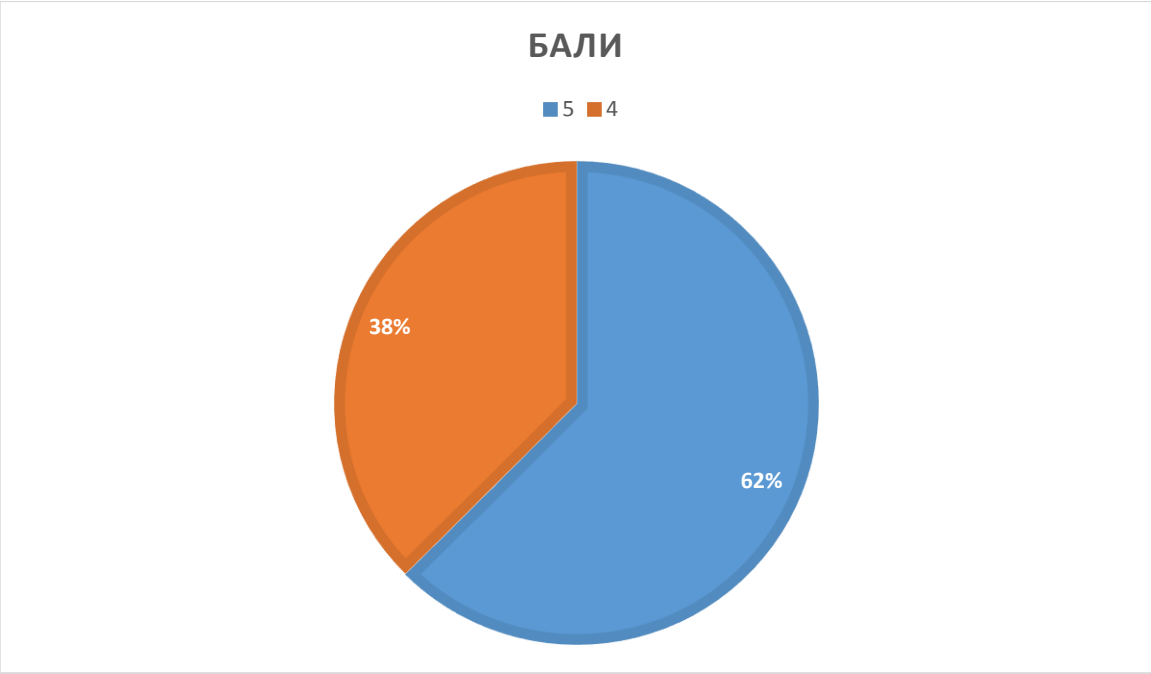


Рис. 2.7. Результати проходження тестування №1

Після проходження студентами теми «Організація циклів та розгалужень» результати тестування були такими: 6 студентів отримали оцінку 5, 6 студентів отримали оцінку4, 3 студенти отримали оцінку 3 та один студент оцінку 2, що показує також досить хороші знання з більш складної теми.

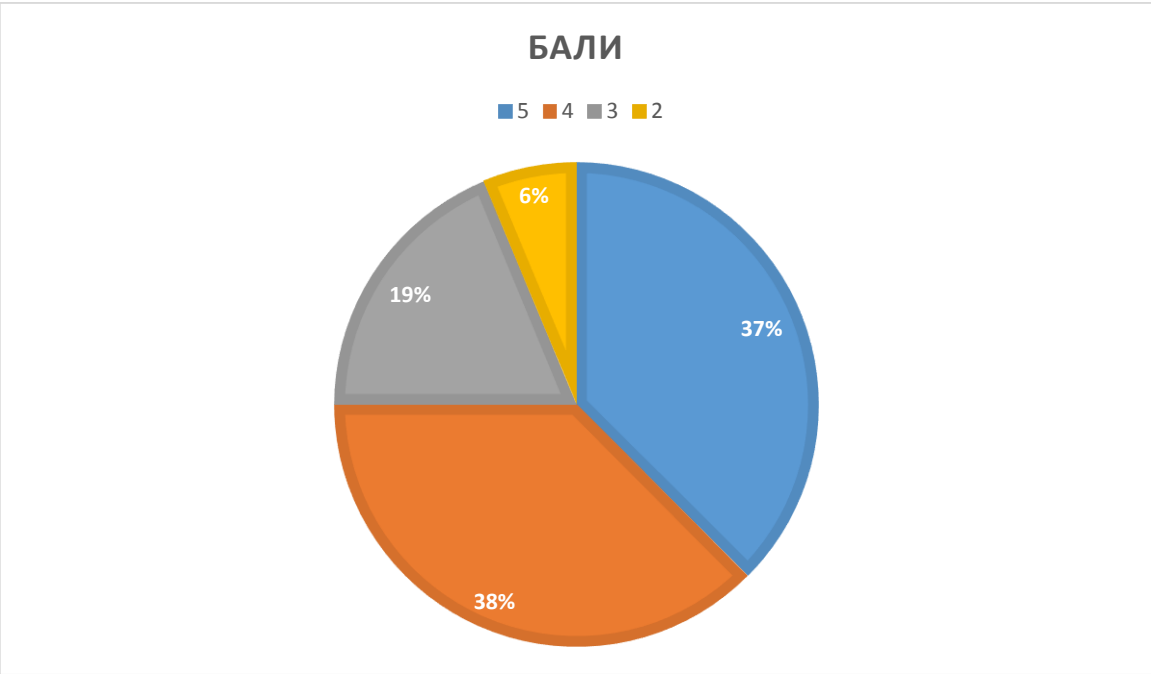


Рис. 2.8. Результати проходження тестування №2

Також ми провели анкетування серед студентів про те, чи допомогла їм запропонована методика краще засвоїти матеріал, що вивчався. Результати опитування зображено на діаграмі:

Рис. 2.9

Таким чином, результати проведених контрольних робіт, тестів та опитувань студентів свідчать про ефективність розробленої методики вивчення програмування мовою Python.

Висновки до розділу 2

У даному розділі розглянуто методику вивчення основ програмування мовою Python майбутніми вчителями математики. Зокрема, розроблено навчально-методичний комплекс з даного розділу, підібрано відповідні методи, форми і засоби навчання. Зокрема, розроблено власний блог «Основи програмування мовою Python», на якому розміщено тексти лекцій, практичних і лабораторних робіт, відео-уроки, тести тощо.

В ході експериментального дослідження проведено підсумковий контроль знань і вмінь студентів групи 2 СОМ з обраних тем. Проведені дослідження й аналіз їх результатів дають нам змогу зробити висновок про ефективність розробленої методики і рекомендувати мову С Python для використання у навчальному процесі під час вивчення майбутніми вчителями математики розділу «Основи програмування» в курсі інформатики.

ЗАГАЛЬНІ ВИСНОВКИ

У ході роботи була досліджена предметна область, розглянуто мови програмування та їх класифікації зроблений висновок, що мова програмування Python , розглянули загальні відомості про мову програмування. Історія її створення, основні типи і структури даних, синтаксис і семантика. Дослідили область застосування мови Python, порівняли середовища програмування та визначили найкращі. Проаналізували роль і місце програмування в курсі інформатики, зробили аналіз робочих програм з інформатики для студентів спеціальностей 014 Середня освіта (Математика) та 111 Математика. Вибрали методи, форми та засоби навчання для розробки навчально-методичного комплексу.

В результаті роботи було розроблено навчально-методичний комплекс вивчення програмування мовою Python в якому вивчаються «Прості типи даних», «Алгоритми» та «Організація циклів та розгалужень», всю інформацію було розміщено в блозі: https://panchenko38.blogspot.com/ Визначили шляхи підвищення пізнавальної мотивації навчальної діяльності студентів.

Виконали експериментальну перевірку ефективності розробленої методики та аналіз результатів дослідження. Отримали позитивні результати знань студентів після проходження нашого комплексу. Результати проведених контрольних робіт, тестів та опитувань студентів свідчать про ефективність розробленої методики вивчення програмування мовою Python. Всі завдання дипломної роботи були виконані в повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Гейн А.Г., Житомирский В.Г., Линецкий Е.В., Сапир М.В., Шолохович В.Ф. «Основы информатики и вычислительной техники»: пробный учебник для 10-11 классов средней школы. // М.: Просвещение. - 1991.
2. Ковтонюк Г. М. До питання використання комп'ютерного тестування у системі контролю якості знань майбутніх учителів / Г. М. Ковтонюк // Сучасні інформаційні технології та інноваційні методики навчання у підготовці фахівців: методологія, теорія, досвід, проблеми. – Випуск 42. – Київ-Вінниця: ТОВ „Планер", 2015. – С. 270-273.
3. Ковтонюк Г. М. До питання формування інформатичної компетентності майбутніх учителів фізико-математичних дисциплін / Г. М. Ковтонюк // Нова педагогічна думка. – 2017, № 3(91). – С. 49-51.
4. Панченко О. В. До питання вивчення основ програмування мовою Python майбутніми вчителями математики / О. Панченко, Г. Ковтонюк // Матеріали II Всеукраїнської науково-практичної інтернет-конференції «Математика та інформатика у вищій школі: виклики сучасності» (Вінниця, 15-16 травня 2019 р.) [Електронне наукове видання]: збірник матеріалів. – Вінниця, 2019. – С. 122-125. – Режим доступу: http://conference.vspu.edu.ua/index.php/MIinHS/MIinHS2019/paper/viewFile/62/62
5. Ковтонюк Г. М. Модель формування професійної готовності майбутніх учителів фізико-математичних дисциплін до організації самостійної пізнавальної діяльності школярів / Г. М. Ковтонюк // Матеріали міжнародної науково-практичної конференції «Проблеми та перспективи фахової підготовки вчителя математики». – Вінниця, 2012. – С. 139-141.
6. Ковтонюк Г.М. Про використання персонального сайту викладача у навчальному процесі // Матеріали Міжнародної науково-методичної Інтернет-конференції «Проблеми вищої математичної освіти: виклики сучасності» (Вінниця, 17-18 травня 2018 р.). – Режим доступу: https://conferences.vntu.edu.ua/index.php/pmovc/pmovc. (0,1 д.а.)

7. Златопольский Д.М. Основы программирования на языке Python. /Д. М. Златопольский / – 2017. – 286 с.

8. Каймин В.А., Щеголев А.Г., Ерохина Е.А., Федюшин Д.П. «Основы информатики и вычислительной техники»: пробное учебное пособие для 10-11 классов средней школы. // М.: Просвещение. - 1989.

9. Калинин И. А., Самылкина Н. Н. Информатика. УМК для старшей школы: 10 - 11 классы (ФГОС). Методическое пособие для учителя. Углублённый уровень // М.: БИНОМ. Лаборатория знаний. - 2013.

10. Калинин И. А., Самылкина Н. Н., Информатика. Программа для старшей школы: 10-11 класс. Углубленный уровень. // М.: БИНОМ. Лаборатория знаний. - 2013.

11. Калинин И. А., Самылкина Н. Н., Информатика. Углублённый уровень: учебник для 10 класса. // М.: БИНОМ. Лаборатория знаний. - 2013.

12. Калинин И. А., Самылкина Н. Н., Информатика. Углублённый уровень: учебник для 11 класса. // М.: БИНОМ. Лаборатория знаний. - 2013.

13. Калинин И. А., Самылкина Н. Н., Бочаров П. В. Информатика. Углубленный уровень: задачник-практикум для 10-11 классов // М.: БИНОМ. Лаборатория знаний. – 2014

14. Колин К. К. Информатика как наука: история и перспективы развития. // Открытое образование. - 2011. - С. 77.

15. Колин К. К. О структуре научных исследований по комплексной проблеме «Информатика» // Сб. н. тр. «Социальная информатика». М.: ВКШ при ЦК ВЛКСМ. - 1990. С. 19-33

16. Курнишенко А.Г., Лебедев Г.В., Сворень Р.А. «Основы информатики и вычислительной техники»: пробный учебник для средних учебных заведений. // М.: Просвещение. - 1990.

17. Лутц М. Изучаем Python. // СПб.: Символ-Плюс. - 2011.

18. Лысенкова С. Н. Методом опережающего обучения. // М.: Просвещение. - 1988. - Т. 125.

19. Монахов В. М. Тридцать лет спустя. // Информатика и образование. - 2015. - №. 7.

20. Новиков Д. А. Статистические методы в педагогических исследованиях (типовые случаи) // М.: Мз-Пресс. - 2004. - С.67

21. Поляков К. Ю., Еремин Е. А. Информатика. Углубленный уровень: учебник для 10 класса: в 2 ч., ч. 2 // М.: БИНОМ. Лаборатория знаний. - 2013.

22. Поляков К. Ю., Еремин Е. А. Информатика. Углубленный уровень: учебник для 11 класса: в 2 ч., ч. 2 // М.: БИНОМ. Лаборатория знаний. – 2013.

23. Семакин И.Г., Шейна Т.Ю., Шестакова Л.В. Информатика. Углубленный уровень: учебник для 10 класса: в 2 ч., ч.1 // М.: БИНОМ. Лаборатория знаний. - 2014.

24. Семакин И.Г., Шейна Т.Ю., Шестакова Л.В. Информатика. Углубленный уровень: учебник для 11 класса: в 2 ч., ч.1 // М.: БИНОМ. Лаборатория знаний. - 2014.

25. Семакин И.Г., Хеннер Е.К. Информатика в профильной школе. // Информатика и образование. - 2010. - №10

26. Ткач С. С. Методические аспекты изучения раздела «Алгоритмизация и программирование» в современном школьном курсе информатики. Электронные ресурсы

27. Ершов А. П. Программирование - вторая грамотность. // Архив академика А.П. Ершова [Электронный ресурс]. - Режим доступа: http://ershov.iis.nsk.su/russian/second_literacy/article

28. Лебедева Т.Н. Становление курса информатики в школьном курсе информатики в школьном образовании в период с 1950 г. до 1990 г. [Электронный ресурс]. –

Режим доступа:http://cyberleninka.ru/article/n/stanovlenie-kursa-informatiki-v-shkolnom- obrazovanii-v-period-s-1950-g-do-1990-g

Додаток А

ЛЕКЦІЯ

ТЕМА: Організація циклів і розгалужень

План

Література

1. Довгалець С. М. Алгоритмічні мови та програмування. Частина 1. Основи інформатики та комп'ютерної техніки. Навчальний посібник / С. М. Довгалець, Р. В. Маслій. – Вінниця : ВНТУ, 2009. – 116 с.

2. Язык программирования Python / Г. Россум, Ф. Л. Дж. Дрейк, Д. С. Откидач та ін. 2001. – 454 с.

3. Бизли Д. Python. Подробный справочник / Д. Бизли. – СПб. : Символ-Плюс, 2010. – 864 с.

4. Хахаев И. А. Python. Практикум по алгоритмизации и программированию на Python / И. А. Хахаев. – М. : Альт Линукс, 2010. – 126 с.

5. Любанович Б. Python. Простой Python. Современный стиль программирования / Б. Любанович. – СПб. : Питер, 2016. – 480 с.

6. Федоров Д. Ю. Основы программирования на примере языка Python : учеб.пособие / Д. Ю. Федоров. – СПб. : Питер, 2016. – 176 с.

7. Лутц М. Изучаем Python, 4-е издание / М. Лутц. – СПб. : Символ-Плюс, 2011. – 1280 с.

1.1. Основні алгоритмічні структури

Основними алгоритмічними структурами є: слідування, розгалуження, цикл.

Слідування – команди виконуються послідовно одна за іншою.

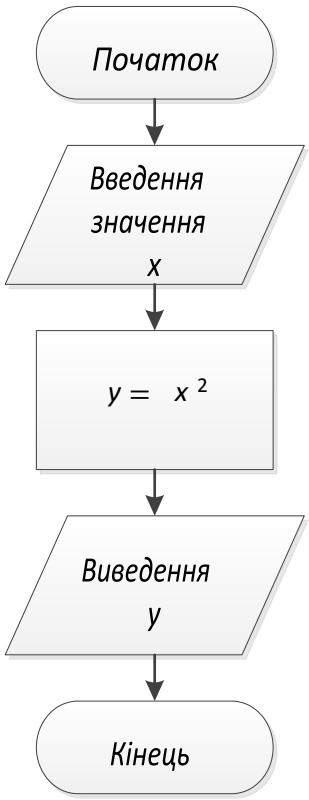


Рис.1 Приклад блок-схеми реалізації лінійного алгоритму

Розгалуження – алгоритм, що містить хоча б одну умову в результаті перевірки якої може виконуватись розділення на декілька паралельних гілок. Кожна з гілок може містити також розділення, послідовні дії або цикли.

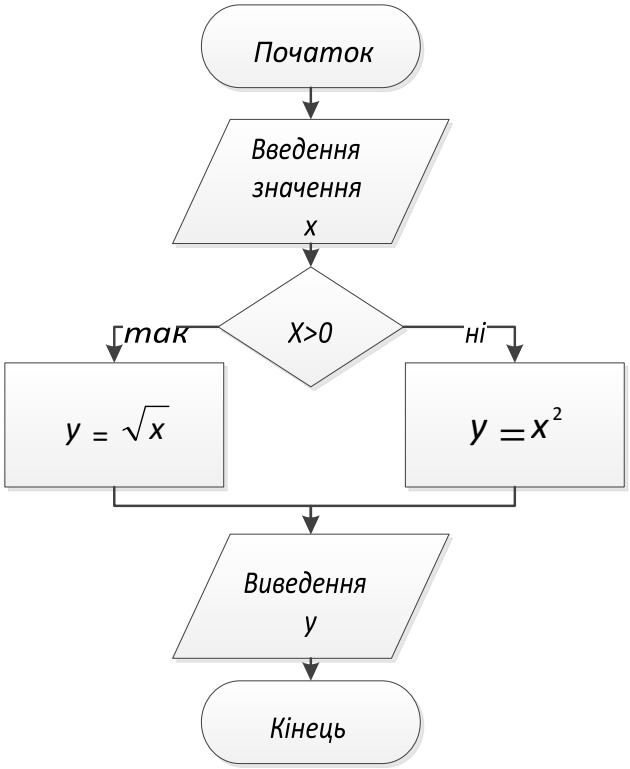
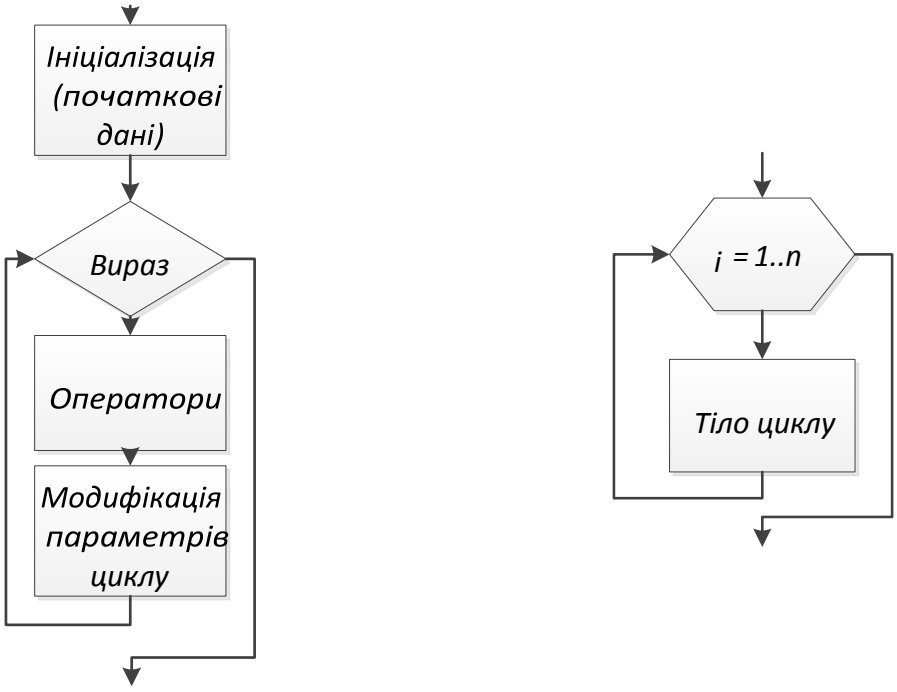


Рис.2 Приклад блок-схеми реалізації алгоритму з розгалуженням

Цикл – інструкції що виконують одну і ту ж послідовність дій поки діє задана умова.



а б

Рис.3 Приклад блок-схеми реалізації циклічного алгоритму (а – цикл з передумовою, б – цикл з лічильником)

2.1. Реалізація алгоритмів з розгалуженням

Хід виконання програми може бути лінійним, тобто таким, коли вирази виконуються, починаючи з першого і закінчуючи останнім, по порядку, не пропускаючи жодного рядка коду. Але частіше буває зовсім не так. При виконанні програмного коду деякі його ділянки можуть бути пропущені.

Припустимо, в реальному житті людина живе за розкладом (можна сказати, розклад – це своєрідний "програмний код", який слід виконати). У її розкладі о 18.00 стоїть похід в басейн. Однак людині надходить інформація, що басейн не працює. Цілком логічно скасувати своє заняття з плавання. Тобто однією з умов відвідування басейну повинно бути його функціонування, інакше повинні виконуватися інші дії.

Схожа не лінійність дій може бути і в комп'ютерній програмі. Частина коду повинна виконуватися лише при певному значенні конкретної умови. Найпростішою в Python для опису розгалуженої структури, де дії виконуються лише у випадку істинності умови, є така конструкція:

if ЛОГІЧНА_УМОВА:
ПОСЛІДОВНІСТЬ_ВИРАЗІВ

Цю конструкцію на блок-схемі можна зобразити:

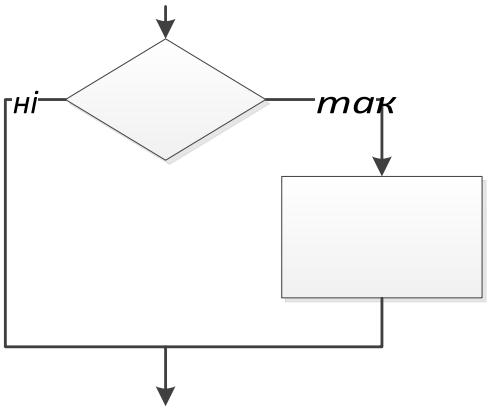


Рис.4 Блок-схема реалізації конструкції if

Першим йде ключове слово ***if*** (англ. "Якщо"); за ним – логічний вираз; потім двокрапка, що позначає кінець заголовка оператора, а після неї – будь-яка послідовність виразів або тіло умовного оператора, яке буде виконуватися в разі, якщо умова в заголовку оператора істинна.

```
x = 2 if
x > 0:
   print("x – додатнє")
if x < 0:    print("x –
від’ємне")
```

Результатом запуску даного коду буде:

```
x – додатнє
```

1. Привласнили значення *2* змінній *x*.
2. Зробили умовне порівняння за допомогою операторів *if*, виконуючи різні фрагменти коду в залежності від значень змінної *x*.
3. Викликали функцію *print()*, щоб вивести текст на екран.

Рядки *if* в Python є операторами, які перевіряють, чи є значення виразу (в даному випадку змінна x) рівним *True*.

print() – це вбудована в Python функція для виведення інформації. Вбудовані функції Python – це іменовані фрагменти коду, які виконують певні операції.

Кожен рядок *print()* відокремлений пробілами під відповідною перевіркою.

У більшості мов програмування символи начебто фігурних дужок (*{*і*}*) або ключові слова *begin* і *end* застосовується для того, щоб розбити код на розділи. У цих мовах хорошим тоном є використання відбиття пробілами, щоб зробити програму більш зрозумілою для себе та інших. Існують навіть інструменти, які допоможуть красиво вибудувати код.

Гвідо ван Росум при розробці Python вирішив, що виділення пробілами буде досить, щоб задати структуру програми і уникнути уведення дужок. Python відрізняється від інших мов тим, що пробіли в ньому використовуються для того, щоб задати структуру програми.

Як правило, використовують чотири пробіли для того, щоб виділити кожен підрозділ, хоча можна використовувати будь-яку кількість пробілів, Python чекає, що всередині одного розділу буде застосовуватися однакова кількість пробілів.

Рекомендований стиль – PEP-8 (http://bit.ly/pep-8) – використовувати чотири пробіли. Не рекомендується застосовувати табуляцію або поєднання табуляцій і пробілів – це заважає підраховувати відступи.

З огляду на це, в конструкції *if* код, який виконується при істинності умови, повинен обов'язково мати відступ вправо. Решта коду (основна програма) повинен мати той же відступ, що і слово *if*.

Зустрічається і більш складна форма розгалуження: *if-else*. Якщо умова при інструкції *if* є хибною, то виконується блок коду при інструкції *else*:

```
if ЛОГІЧНА_УМОВА:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1
else:
   ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2
```

Цю конструкцію на блок-схемі можна зобразити:

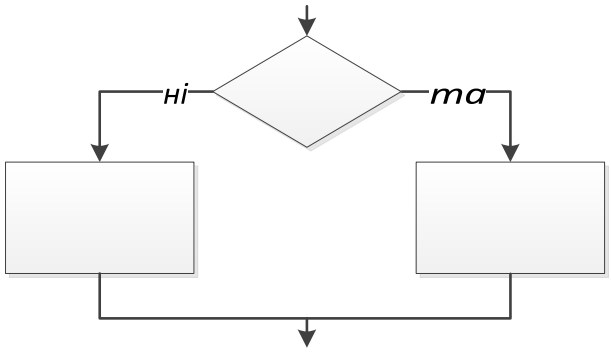


Рис.5 Блок-схема реалізації конструкції if-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Якщо умова не виконується потік виконання переходить до рядка, який йде після *else*.

```
x = int(input("Введіть x=
")) if x
> 0:
    y=x**0.5
else:
    y=x**2
print("y =", y)
```

Отримаємо:

```
Введіть x= 9
y = 3.0
```

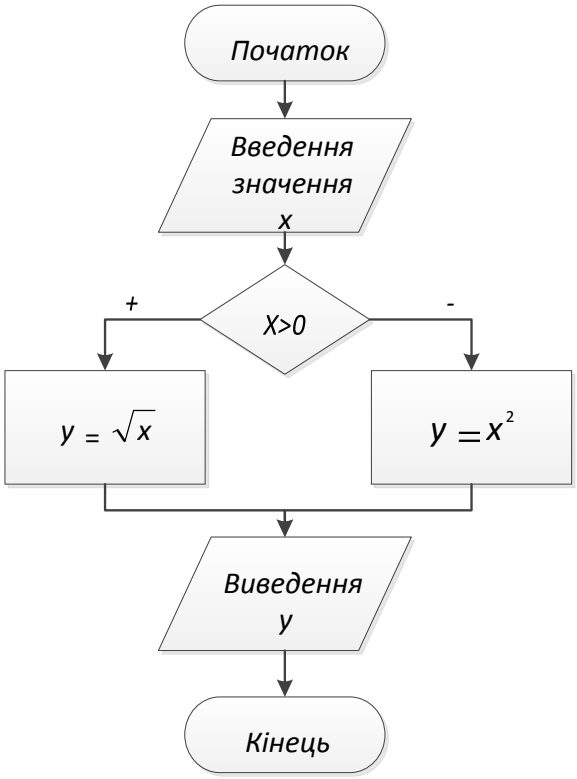


Рис.6 Приклад блокконструкції if-схеми реалізації -else

Альтернативні гілки програми

Логіка програми що виконується може бути складнішою, ніж вибір однієї з двох гілок.

Умовний оператор *if* має розширений формат, що дозволяє перевіряти кілька незалежних одна від одної умов і виконувати один з блоків, поставлених у відповідність з цими умовами. У загальному вигляді оператор виглядає так:

if ЛОГІЧНА_УМОВА_1:

ПОСЛІДОВНІСТЬ_ВИРАЗІВ_1

elif ЛОГІЧНА_УМОВА_2:

ПОСЛІДОВНІСТЬ_ВИРАЗІВ_2

elif ЛОГІЧНА_УМОВА_3:

ПОСЛІДОВНІСТЬ_ВИРАЗІВ_3

... else:

ПОСЛІДОВНІСТЬ_ВИРАЗІВ_N

Цю конструкцію на блок-схемі можна зобразити:

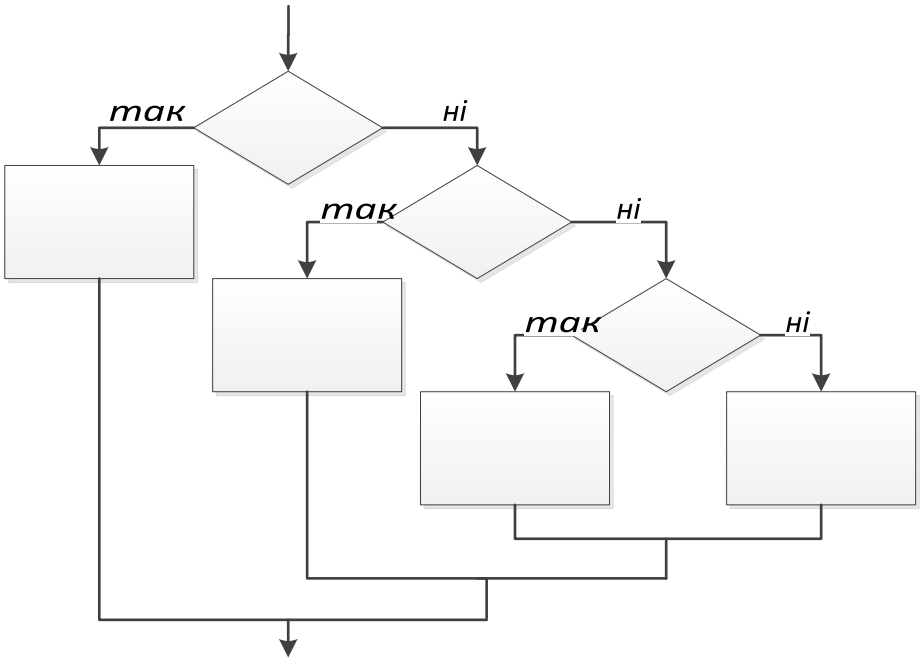


Рис.7 Блок-схема реалізації конструкції if-elif-else

Працює ця конструкція наступним чином. Спочатку перевіряється перша умова і, якщо вона істинна, то виконується перша послідовність виразів. Після цього потік виконання переходить до рядку, який йде після умовного оператора (тобто за послідовністю виразів *N*). Якщо перша умова рівна *False*, то перевіряється друга умова (наступна після *elif*), і в разі його істинності виконується послідовність 2, а потім знову потік виконання переходить до рядка, наступного за оператором умови. Аналогічно перевіряються всі інші умови. До гілки програми *else* потік виконання доходить тільки в тому випадку, якщо не виконується жодна з умов.

Ключове слово *elif* походить від англ. "*Else if*" – "інакше якщо". Тобто умова, яка слідує після нього перевіряється тільки тоді, коли всі попередні умови хибні.

```
if not True:
print("1") elif not
(1+1==3):
   print("2")
else:
print("3")
```

Результатом запуску даного коду буде:

3

Оператор pass

В процесі роботи над програмою слід намагатися після кожної зміни мати працюючу програму, але іноді не завжди відразу відомо, що необхідно виконати якщо умова приймає істинне значення, а що в протилежному випадку. В Python інструкції з розгалуженням, або цикли, або функції з порожнім тілом заборонені, тому в якості тіла використовується "порожній оператор" ***pass***.

Припустимо, що заплановано використання умовного оператора з декількома умовами, але встигнуто написати тільки один з блоків умовного оператора. При цьому постає питання, як її налагодити, якщо програма не виконується через синтаксичну помилку.

```
if not True:
print("1") elif not
(1+1==3): elif not
(1+1==4): elif not
(1+1==5):
```

Блоки для випадків, коли значення not (1+1==3), not (1+1==4), not (1+1==5), ще не написані, тому програма не виконується через помилку SyntaxError: expected an indented block.

Ключове слово *pass* можна вставити на місце відсутнього блоку:

```
if not True:
print("1") elif not
(1+1==3):    pass
elif not (1+1==4):
```

pass elif not
(1+1==5): pass

3.1. Реалізація циклічних алгоритмів

У реальному житті ми досить часто зустрічаємося з циклами. У комп'ютерних програмах поряд з інструкціями розгалуження (тобто вибором шляху дії) також існують інструкції циклів (повторення дії). Якби інструкцій циклу не існувало, довелося б багато разів вставляти в програму один і той же код поспіль стільки раз, скільки потрібно виконати однакову послідовність дій.

Цикли – це інструкції, які виконують одну й ту ж саму послідовність дій, поки діє задана умова.

Кожен циклічний оператор має тіло циклу – якийсь блок коду, який інтерпретатор буде повторювати поки умова повторення циклу буде залишатися істинною.

В програмуванні розрізняють такі види циклів:

- Цикл з передумовою – виконує дії поки умова є істинною (while).
- Цикл з післяумовою – спочатку виконуються команди, а потім перевіряється умова (do…while).
- Цикл з лічильником – виконується задану кількість разів (for).
- Сумісний цикл – виконує команди для кожного елементу із заданого набору значень (for…which).

В Python присутні лише цикл з передумовою (while) та цикл for, що поєднує в собі два види – цикл з лічильником та сумісний цикл.

Оператор циклу while

Універсальним організатором циклу в мові програмування Python є конструкція *while* [5, 6, 7, 8, 9, 11]. Слово "while" з англійської мови перекладається як "поки" ("поки логічний вираз повертає істину, виконувати певні операції"). Конструкцію *while* на мові Python можна описати наступною схемою:

while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:

ТІЛО_ЦИКЛУ

Дуже схоже на оператор умови – тільки тут використовується інше ключове слово: *while* (англ. "Поки").

Дану конструкцію як правило використовують при повторі уведення даних користувачем, поки не буде отримано коректне значення:

```
correct_choice = False while
not correct_choice:
   choice = input("Введіть число 1 або
2:")    if choice == "1" or choice == "2":
correct_choice = True    else:
      print ("Не правильно введено число, повторіть введеня")
```

Результатом запуску даного коду буде:

```
Введіть число 1 або 2:3
Не правильно введено число, повторіть введеня. Введіть число 1
або 2:4
Не правильно введено число, повторіть введеня
Введіть число 1 або 2:1
>>>
```

1. Визначили логічну змінну *correct_choice*, присвоївши їй значення *False*.
2. Оператор циклу перевіряє умову *not correct_choice*: заперечення False – істина. Тому починається виконання тіла циклу: виводиться запрошення *"Enter your choice, please (1 or 2):"* і очікується уведення користувача.
3. Після натискання клавіші Enter введене значення порівнюється з рядками *"1"* і *"2"*, і якщо воно дорівнює одній з цих значень, то змінній *correct_choice* присвоюється значення *True*. В іншому випадку програма виводить повідомлення *"Не правильно введено число, повторіть введеня"*.
4. Оператор циклу знову перевіряє умову і якщо вона як і раніше істинна, то тіло циклу повторюється знову, інакше потік виконання переходить до наступного оператору, і інтерпретатор

виходить з циклу і виконання програми припиняється або виконуються дії що прописані поза тілом циклу.

Ще один варіант використання оператора циклу – обчислення формул із змінним параметром.

$$\sum_{i=1}^{n} i^3 = 1^3 + \cdots + n^3$$

В даному випадку, параметром, що змінюються є *i*, причому *i* послідовно приймає значення в діапазоні від *1* до *n*.

За допомогою оператора циклу *while* рішення буде виглядати так:

```
n = int(input("Введіть n: "))
sum = 0 i = 1
while i <= n:
sum += i**3
   i += 1
print ("sum = ", sum)
```

Результатом запуску даного коду буде:

```
Введіть n: 5 sum
=  225
```

1. Необхідно ввести *n* – граничне значення *i*.
2. Ініціалізація змінної *sum* – в ній буде зберігатися результат, початкове значення – *0*.
3. Ініціалізація змінної *i* (лічильника *i*) – за умовою, початкове значення – *1*.
4. Починається цикл, який виконується, поки *i* <= *n*.
5. У тілі циклу в змінну *sum* записується сума значення з цієї змінної, отриманої на попередньому кроці, і значення *i*, піднесеної в куб.
6. Лічильнику *i* присвоюється наступне значення.
7. Після завершення циклу виводиться значення *sum* після виконання останнього кроку.

Наступне значення лічильника отримують додаванням до його поточного значення кроку циклу (в даному випадку крок циклу дорівнює *1*). Крок циклу при необхідності може бути від’ємним, і

навіть дробовим. Крім того, крок циклу може змінюватися на кожній ітерації (тобто при кожному повторенні тіла циклу).

Нескінченні цикли

Іноді можна зіткнулися з проблемою нескінченного повторення блоку коду, що виникає, наприклад, через семантичну помилку в програмі:

```
i = 0 while i
< 10:     print
(i)
```

Такий цикл буде виконуватися нескінченно, тому що умова *i <10* завжди буде істинною, адже значення змінної *i* не змінюється: така програма буде виводити нескінченну послідовність нулів.

У циклів немає обмеження кількості повторень тіла циклу, тому програма з нескінченним циклом буде працювати безперервно, поки не буде зроблено аварійної зупинки натисканням комбінації клавіш Ctrl+C, не зупинимо відповідний процес засобами операційної системи, або не будуть вичерпані доступні ресурси комп'ютера (наприклад, може бути досягнуто максимально допустимої кількості відкритих файлів), чи не виникне виняток. У таких ситуаціях кажуть, що програма зациклилася.

Знайти зациклення в програмі іноді буває не так-то просто, адже в реальних програмах зазвичай використовується багато циклів, кожен з яких потенційно може виконуватися нескінченно.

Проте, відсутність обмеження на кількість повторів тіла циклу дає нові можливості. Багато програм представляють собою цикл, в тілі якого проводиться відстеження та оброблення дій користувачів або запитів з інших програмних і автоматизованих систем. При цьому такі програми можуть працювати без перебоїв дуже тривалі терміни, іноді роками.

Цикли – це дуже потужний засіб реалізації, але вони вимагають деякої уважності.

Якщо необхідно, щоб цикл виконувався до тих пір, поки щось не станеться, але точно не відомо, коли ця подія трапиться, можна

скористатися нескінченним циклом, що містить оператор ***break***.

```
i = 1

while True:
    if i > 5:
        break
    print (i)
     i+= 1
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

1. Виведемо 5 чисел починаючи з *1*. Змінній *i* привласнено початкове значення *1*.
2. Найпростішою умовою для нескінченного циклу є значення True.
3. Виконується перевірка значення змінної *i* зі значенням *5*. Якщо воно співпадає або є більшим, то виконується оператор *break* що переривається виконання циклу.
4. Виводиться на екран значення числа (виконується якщо не спрацював оператор *break).*
5. Збільшується значення змінної *i* на *1*

Іноді потрібно не переривати весь цикл, а лише пропустити по якійсь причині одну ітерацію. ***Continue*** дозволяє перейти до наступної ітерації циклу до завершуючи виконання всіх виразів всередині циклу.

Розглянемо приклад: виводяться на екран цілі числа в діапазоні від 1 до 10, крім чисел 3, 4, 5.

```
i=0

while i<=10:
   i+=1
```

```
if 3<=i<=5:
    continue    print(i)
```

Результатом запуску даного коду буде:

```
1
2
6
7
8
9
10
11
```

Альтернативна гілка else

Мова Python дозволяє використовувати розширений варіант оператора циклу:

while УМОВА_ПОВТОРЕННЯ_ЦИКЛУ:
ТІЛО_ЦИКЛУ
else:
АЛЬТЕРНАТИВНА_ГІЛКА_ЦИКЛУ

Поки виконується умова повторення тіла циклу, оператор *while* працює так само, як і в звичайному варіанті, але як тільки умова повторення перестає виконуватися, потік виконання направляється по альтернативній гілці *else* – так само, як в умовному операторі *if*, вона виконається всього один раз.

```
i = 0
while i<3:
    print (i)
    i += 1
else:
    print ("кінець циклу")
```

Результатом запуску даного коду буде:

```
0
1
2 кінець
циклу
```

Цикл for

В Python ітератори часто використовуються оскільки вони дозволяють проходити структури даних, не знаючи, наскільки ці структури великі і як реалізовані. Можливо пройти по послідовності таким чином:

```
numbers = [1, 2, 3, 4, 5]
i = 0

while i <=len(numbers)-1:
    print(numbers[i])
     i+=1
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

Однак існує більш характерний для Python спосіб вирішення цього завдання:

```
for element in numbers:
print(element)
```

Результатом запуску даного коду буде:

```
1
2
3
4
5
```

Списки подібні до *numbers* є одними з ітераційних об’єктів в Python поряд з рядками, кортежами, словниками та деякими іншими елементами.

Ітераційні об’єкти – це ті, вміст яких можна перебрати в циклі. Ітерування по кортежу або списку повертає один елемент за раз. Ітерування по рядку повертає один символ за раз:

```
word = 'cat' for
letter in word:
    print(letter)
```

Результатом запуску даного коду буде:

```
c
a
t
```

Цикл *for* дозволяє перебирати всі елементи вказаної послідовності (список, кортеж, рядок). Цикл спрацює рівно стільки разів, скільки елементів знаходиться в послідовності.

for ЗМІННА in ПОСЛІДОВНІСТЬ:
ТІЛО_ЦИКЛУ

Ім’я змінної в яку на кожному кроці буде розміщено елемент послідовності обирається довільно.

Оператори *break* та *continue* працюють так само як і в циклі *while*. Блок *else* дозволяє перевірити чи виконався цикл *for* повністю якщо ключове слово *break* не було викликане, то буде виконаний блок *else.* Це корисно, якщо потрібно переконатися в тому, що попередній цикл виконався повністю, замість того щоб рано перерватися:

```
numbers = [] for
number in numbers:
    print('Цей список має деякий елемент',number)
    break
else:
# Відсутність переривання означає, що елемент відсутній

    print('Елементів немає в списку, чи не так?')
```

Результатом запуску даного коду буде:

```
'Елементів немає в списку, чи не так?'
```

В циклах використання блоку *else* може здатися нелогічним. Можна розглядати цикл як пошук чогось, в такому випадку *else* буде викликатися, якщо нічого не було знайдено.

Ітерування за кількома послідовностям за допомогою функції zip()

Щоб паралельно ітеруватися за кількома послідовностям одночасно можна використати функцію *zip()*:

```
days = ['Monday', 'Tuesday', 'Wednesday']
fruits = ['banana', 'orange'] drinks = ['coffee', 'tea', 'beer']
for day, fruit, drink in zip(days, fruits, drinks):
    print(day, ": drink", drink, "eat", fruit)
```

Результатом запуску даного коду буде:

```
Monday : drink coffee eat banana
Tuesday : drink tea eat orange
```

Функція *zip()* припиняє свою роботу, коли виконується найкоротша послідовність. Один зі списків (fruits) виявився коротшим за інші, тому результат було виведено лише для двох елементів.

Функція *zip()* – ітератор, який повертає кортежі, що складаються з відповідних елементів аргументів-послідовностей.

Генерація числових послідовностей за допомогою функції range()

Функція ***range()*** повертає послідовність чисел в заданому діапазоні без необхідності створювати і зберігати велику структуру даних на зразок списку або кортежу.

Це дозволяє створювати великі діапазони, не використавши всю пам’ять комп’ютера і не обірвавши виконання програми.

range (start, end, step)

Якщо опустити значення *start*, діапазон почнеться з 0. Необхідною є лише значення *end*, що визначає останнє значення, яке буде створено прямо перед зупинкою функції. Значення step по замовчуванню дорівнює 1, але можна змінити його на -1 [5 - 13].

Якщо просто викликати функцію *range()*, то результату не буде отримано.

```
>>> range(0,10,1) range(0,
10)
```

Як і zip (), функція *range ()* повертає ітераційний об’єкт, тому потрібно пройти за значеннями за допомогою конструкції *for ... in* або перетворити об’єкт в послідовність (список, кортеж та ін.).

```
for x in range(0, 3):
    print(x)
```

Результатом запуску даного коду буде:

```
0
1
2
```

```
for x in range(2, -1, -1):
    print(x)
```

Результатом запуску даного коду буде:

```
2
1
0
```

Підходи до створення списків Створити список цілих чисел можна декількома способами.

Можна додавати елементи до списку по одному за раз використовуючи функцію *append()*:

```
number_list = []
print (number_list)

number_list.append(1)
print (number_list)

number_list.append(2)
number_list.append(3)
number_list.append(4)
number_list.append(5) print
(number_list)
```

Результатом запуску даного коду буде:

```
[]
```

[1]
[1, 2, 3, 4, 5]

Можна використати ітератор та функцію *range()*:

```
number_list = [] for
number in range(1, 6):
    number_list.append(number) print
(number_list)
```

Отримаємо:

[1, 2, 3, 4, 5]

Або ж перетворити в список сам результат роботи функції *range()*:

```
number_list = list(range(1, 6)) print
(number_list)
```

Отримаємо:

[1, 2, 3, 4, 5]

Однак можна створити список за допомогою включення списку.

Спискове включення

Включення – це компактний спосіб створити структуру даних з одного або більше ітераторів. Включення дозволяють вам об'єднувати цикли і умовні перевірки, не використовуючи при цьому громіздкий синтаксис.

Найпростіша форма такого включення виглядає так:

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ'ЄКТ]

```
number_list = [number for number in range(1,6)] print (number_list)
```

Отримаємо:

[1, 2, 3, 4, 5]

У першому рядку потрібно, щоб перша змінна *number* сформувала значення для списку: слід розмістити результат роботи циклу в змінну *number_list*. Друга змінна *number* є частиною циклу *for*. Щоб показати, що перша змінна *number* є виразом, спробуємо такий варіант:

```
number_list = [number-1 for number in range(1,6)]
    print (number_list)
```

Отримаємо:

[0, 1, 2, 3, 4]

Включення списку може містити умовний вираз:

[ВИРАЗ for ЕЛЕМЕНТ in ІТЕРАЦІЙНИЙ_ОБ'ЄКТ if УМОВА]

Наступне включення створює список, що складається тільки з парних чисел, розташованих в діапазоні від 1 до 5 (вираз *number% 2* має значення *True* для парних чисел і *False* для непарних):

```
number_list = [number for number in
range(1,6) if number % 2 == 1]
  print (number_list)
```

Отримаємо:

[1, 3, 5]

Вираз може містити будь-що. Для створення списку з елементів, що будуть вводитися з клавіатури:

```
number_list = [int(input('введіть число: ')) for number in
range(int(input('введіть кількість елементів: ')))]
   print (number_list)
```

Отримаємо:

введіть кількість лементів:5
введіть число: 1

введіть число: 0

введіть число: 5

введіть число: 1

введіть число: 3 [1, 0, 5, 1,3]

Вкладені списки

Списки можуть містити елементи різних типів, включаючи інші списки. ***Вкладеними*** називаються списки, які є елементами іншого списку.

Вкладені списки зазвичай використовуються для подання матриць. Змінною *list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]* описано матрицю:

1 2 3
4 5 6
7 8 9

В змінній *list_list* зберігається список з трьома елементами, в кожному з яких зберігається рядок матриці теж у вигляді списків. Витягти рядок можна за допомогою оператора індексування:

list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] print(list_list)

print(list_list [0])

Отримаємо:

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]

Щоб звернутися або отримати елемент матриці (вкладеного списку) необхідно вказати два індекси:

list_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print(list_list)

print(list_list [0]) print(list_list
[0][1])

Отримаємо:

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[1, 2, 3]
2

Оператори індексування виконуються зліва на право. Індекс [0] посилається на перший елемент списку list_list, а [1] – на другий елемент внутрішнього списку. Перший індекс визначає

номер рядка, а другий – номер елемента (тобто номер стовпчика) в матриці.

Вкладені цикли

Цикли можуть бути вкладені один в одного. При цьому цикли можуть використовувати різні змінні-лічильники.

Найпростіше застосування вкладених операторів циклу – побудова двовимірних таблиць (матриць, масивів), наприклад:

```
i= 1

while i <= 10:
    j= 1
    while j <= 10:
      print (i * j,end="\t")
       j += 1     print ()
       i += 1
```

Цикл, перевіряючий умову *i<=10,* відповідає за повторення рядків таблиці. У його тілі виконується другий цикл, який виводить добуток *i* і *j* 10 разів поспіль, розділяючи знаками табуляції отримані результати (*end="\t"*).

Функція *print()* без параметра виконує перехід на новий рядок.

В результат виконання даної програми отримаємо наступну таблицю:

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Перший стовпець і перший рядок містять числа від 1 до 10.

На перетині *i*-того рядка і *j*-того стовпця знаходиться добуток *i* і *j*. Отримуємо таблицю множення.

Якщо додати ще один цикл, то отримаємо тривимірну таблицю добутків трьох чисел. Таким чином вкладені цикли дозволяють будувати таблиці будь-якої розмірності, що дуже часто застосовується в складних наукових розрахунках.

```
outer = [1, 2, 3]
inner = [4, 5, 6]
for i in outer:
    for j in inner:
        print ('i=', i, 'j=', j)
```

Результатом запуску даного коду буде:

```
i= 1 j= 4 i=
1 j= 5 i= 1
j= 6 i= 2 j=
4 i= 2 j= 5
i= 2 j= 6 i=
3 j= 4 i= 3
j= 5 i= 3 j=
6
```

Цикл *for* спочатку просувається по елементах зовнішнього циклу (фіксуємо *i=1*), потім переходить до вкладеного циклу (змінна *j*) і проходимо по всіх елементах вкладеного списку. Далі повертаємося до зовнішнього циклу (фіксуємо значення *i=2*) і знову проходимо по всіх елементах вкладеного списку. Так повторюємо до тих пір, поки не закінчаться елементи в зовнішньому списку:

Даний прийом активно використовується при роботі з вкладеними списками.

Спочатку приклад з одним циклом *for*:

```
lst = [[1, 2, 3], [4,5,6]]
for i in lst:

  print (i)
```

Результатом запуску даного коду буде:

```
[1, 2, 3]
```

[4, 5, 6]

У прикладі за допомогою циклу *for* перебираються всі елементи списку, які також є списками.

Якщо необхідно дістатися до елементів вкладених списків, то доведеться використовувати вкладений цикл *for*:

```
lst = [[1, 2, 3], [4, 5, 6]]
for i in lst:           # Цикл за елементами зовнішнього списку
    print()
    for j in i:

        print (j, end="")
```

Результатом запуску даного коду буде:

```
123
456
```

Додаток Б

ПРАКТИЧНА РОБОТА №1

ТЕМА: Цикли
МЕТА: Практичне опрацювання та програмна реалізація циклів з умовою які використовуються у програмуванні на мові Python

Завдання 1

Самостійно виконати наступні завдання із використанням циклу з умовою.

1. Розробіть код обчислення суми для чисел: 2, 7, 21, 9, 33, 13.

2. Розробіть код, обчислення суми непарних чисел, що більші 7, але менші 25.

3. Розробіть код обчислення суми чисел натурального ряду, максимальне значення якого не перевищує 7.

Завдання 2

1. Знайти суму n членів ряду, заданого за варіантом. $\sum_{i=1}^{n} \frac{\overline{2i}}{\overline{i+5}-\overline{i}}$
2. Скласти програму яка за введеними з клавіатури значеннями змінних, виконає необхідні розрахунки.

- Знайти суму цифр заданого натурального числа n.
- Дано натуральне число. Підрахувати загальну кількість його дільників.

Онлайн компілятор: https://www.onlinegdb.com/online_python_compiler

ПРАКТИЧНА РОБОТА №2

Тема Алгоритмічні структури циклів з параметрами

Мета: Практичне опрацювання та програмна реалізація циклів з параметрами які використовуються у програмуванні на мові Python

Завдання 1. Вводити з клавіатури числа, поки не введемо нуль

*розв'язок задачі***:**

```
a=int(input("Введіть число"))
while (a!=0):
    a=int(input("Введіть число"))
print ("The End")
```

Завдання 2. Порахувати, скільки чисел генерує комп'ютер, поки не отримає випадкове число 100 в діапазоні від 1 до 100

розмістити рядки коду в правильному порядку:

a=random.randint(1,100), print ("K=",k), print(a, end=' '), while (a!=100):, k=0, k=k +1, a=random.randint(1,100), import random.

Завдання 3. Порахувати, скільки парних і непарних чисел генерує комп'ютер, поки не отримає випадкове число 100 в діапазоні від 1 до 100.

Завдання 4. Порахувати середнє арифметичне отриманих чисел

Завдання 5. Вгадуємо число. Комп'ютер загадує число від 1 до 10. Нам потрібно його вгадати

дописати пропущені рядки коду:

```
import random
print("Я загадую число від ___ до ____")
a=_____________________
print("Ваша версія=",end=' ')
b=_____________________
while (_______):
    if (________):
        print("Мало")
    else:
        print("_________")
    print("Ваша версія=",end=' ')
    b=_____________
```

print ("Вгадано!")

Додаток В

Лабораторна робота №1

Тема: Проектування програм циклічних алгоритмів

Мета роботи: ознайомитися з алгоритмами циклічної структури, їх реалізацією на мові Python; вивчити формат оператору циклу *while*.

Завдання

1. Вивчити теоретичні основи написання алгоритмів циклічної структури. Опрацювати приклади.
2. Побудувати блок-схему алгоритму вирішення завдання 1.
3. Відповідно до свого варіанту

- за допомогою формул описати варіанти виконання необхідний дій;
- написати програму, яка розв‘язує завдання, реалізуючи обробку помилок
- організувати введення даних з клавіатури, виведення у консоль рядків або списків.

4. Скласти звіт і захистити його по роботі.

Захист роботи включає в себе демонстрацію працездатності програми на різних вхідних даних, демонстрацію трасировки виконання програми.

Варіанти

Завдання 1. Скласти програму обчислення значення функції на відрізку $[x_0, x_k]$ з кроком h за допомогою циклу *while*.

№	$f(x)$	h; $[a; b]$
1	2	3
1	$y=\ln(x)$	h=0.1; a=1; b=1.5
2	$y=1+\ln^2(x)$	h=0.1; a=0.4; b=1.0
3	$y=1+e^x$	h=0.01; a=0.5; b=0.6
4	$y= e^{x^2} / 2$	h=0.2; a=2; b=3
5	$y= \cos(x) \cdot e^{-x}$	h=0.2; a=1; b=2
6	$y=1/(1 + e^{-x})$	h=0.2; a=3; b=4
7	$y= \sin(x) \cdot \sinh(x)$	h=1; a=1; b=5
8	$y= 0.5 + \sinh^2(x)$	h=0.2; a=2; b=3
9	$y= \overline{x} \cdot \cosh(x)$	h=0.2; a=3; b=4

$\sqrt{}$

1	2	3
10	$y=1/(1+\cosh^2(x))$	h=0.5; a=2; b=4
11	$y=\overline{x}\cdot\sinh(x)$	h=1; a=1; b=5
12	$y=e^{-x}\cdot\cosh(x)$	h=1; a=1; b=4
13	$y=\ln(x^2)$	h=0.1; a=1; b=1.4
14	$y=x+\ln(x)$	h=1; a=1; b=5
15	$y=1/(1+\sin(x))$	h=π/10; a==π/6; b==π/3
16	$y=\sin(x)+\overline{x}$	h=π/10; a==π/6; b==π/4
17	$y=\sqrt{x}\cdot(1-\cos(x))$	h=0.1; a=0.4; b=0.8
18	$y=e^{x+3}\sin(x)$	h=0.5; a=0; b=2
19	$y=\cos(x)\cdot\cosh(x)$	h=1; a=1; b=5
20	$y=e^{x+1}\cdot\sqrt{\sinh(x)}$	h=1; a=1; b=4
21	$y=10^{-2}(5+4x)-e^{x_3+4}$	h=0.1; a=-3.4; b=-1.4
22	$y=4x^3+2^{5/4}xe^{-x}$	h=1.01; a=2.4; b=10.4
23	$y=9(x^3+3.2)\cdot\mathrm{tg}(x)$	h=0.2; a=1; b=2.4
24	$y=1.2e^{x^2}+x$	h=-0.05; a=-0.75; b=-1.5
25	$y=x^{2_2}+\cos(2^{3/4}+x^{3/2})$	h==π/3; a=14; b=19
26	$y=(x^{5/2}-0.8)\cdot\ln(x^2+12.7)$	h=0.3; a=0.25; b=5
27	$y=0.8\cdot10^{-5}(x^3+6.7)^{7/6}$	h=0.1; a=-0.5; b=0.4
28	$y=0.4+x^{2/3}\cos(x+e^x)$	h=π/10; a=5.6; b=15.4

Контрольні запитання

1. Що таке цикл, для чого його використовують?
2. Як описується та виконується циклічна інструкція while?
3. Як можна організувати нескінченні цикли? Наведіть декілька прикладів і поясніть їх.
4. Як можна вийти з нескінченних циклів? Що відбувається при запуску нескінченного циклу?
5. Чи може оператор циклу не мати тіла? Чому?
6. Для чого служать оператори переривання *break* та *continue*?

Лабораторна робота №2

ТЕМА: Прості типи даних. Проектування програм циклічних алгоритмів

МЕТА: Ознайомитися з алгоритмами циклічної структури, їх реалізацією на мові Python; вивчити формат оператору циклу while.

Теоретичні відомості

Інструкція while. Алгоритм, в якому передбачено неодноразове виконання певної послідовності дій, називається алгоритмом циклічної структури або циклом. Цикл дозволяє істотно скоротити розмір запису алгоритму, зобразити його компактно шляхом відповідної організації пропонуємих дій. Повторювати певні дії має сенс при різних значеннях параметрів, які змінюються. Такі параметри називаються параметрами циклу. Блок повторюваних операторів називають тілом циклу (це послідовність дій, які виконується багаторазово).

Циклічний процес називається ітераційним, якщо заздалегідь невідома кількість повторень циклу, а кінець обчислення визначається при досягненні деякою величиною заздалегідь заданої точності обчислення.

Для запису ітераційних процесів в Python використовують лише один тип операторів циклу while - оператор з попередньою умовою (передумовою). Для всіх операторів циклу характерні такі особливості:

1. Повторювані обчислення записуються лише один раз.
2. Вхід в цикл можливий тільки через його початок.
3. Змінні оператора циклу повинні бути визначені до входу в цикл.
4. Потрібно передбачити вихід з циклу. Якщо цього не зробити, то обчислення будуть тривати нескінченно довго.

Нескінченний цикл – це циклічна ділянка в програмі, в якій не передбачені засоби виходу з циклу при досягненні деякого умови.

Інструкція while організує цикл з передумовою (перевірка виконується перед початком чергової ітерації), складається з рядка заголовка з умовним виразом, тіла циклу, що містить одну або більше інструкцій з відступами, і необов'язкової частини else, яка виконується, коли управління передається за

межі циклу без використання інструкції break. Інтерпретатор продовжує обчислювати умовний вираз в рядку заголовка і виконувати вкладені інструкції в тілі циклу, поки умовний вираз не поверне значення «хибність»:

Приклад програми:

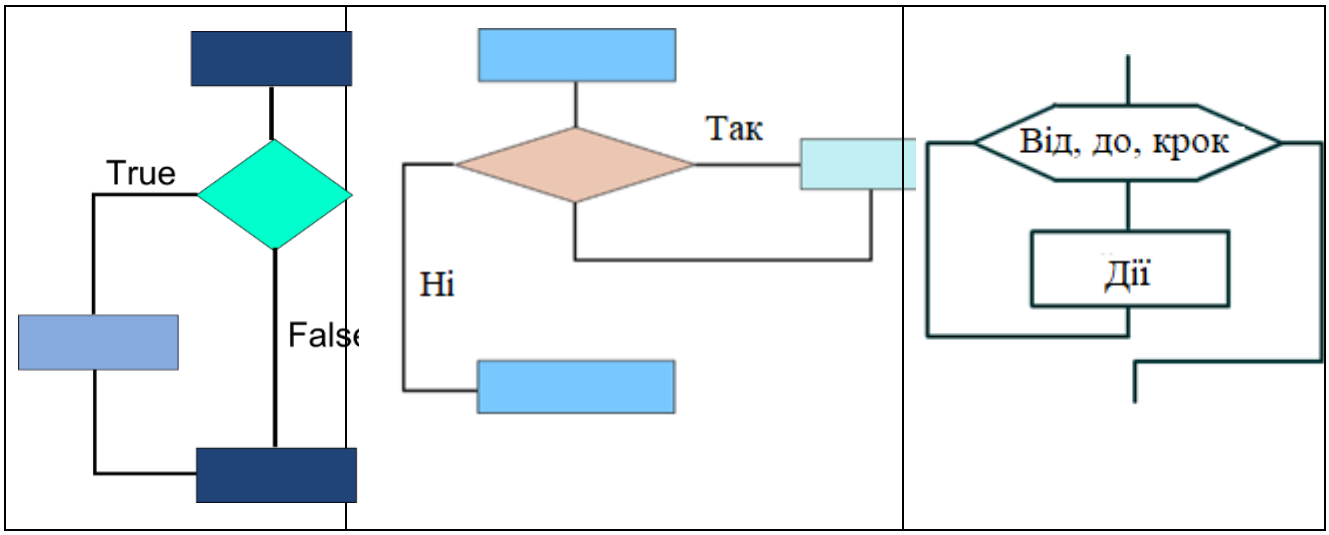
```
 while
answer = "no"
while answer != "yes":
answer = raw_input("Ви любите Python? ")
if answer == "yes":
        print "Відмінно!"
else:
        print "Неправильна відповідь! Спробуйте знову."
```

Інша версія того ж самого:

```
while
answer = raw_input("Ви любите Python? ")
while answer != "yes":
print "Неправильна відповідь! Спробуйте знову."
answer = raw_input("Ви любите Python? ")
else:
print "Відмінно!"
```

Програма зупиняється, якщо користувач правильно відповідає " *yes* ".

Базові алгоритмічні конструкції:



1- Розгалуження 2 – Цикл while 3 – Цикл for

Хід роботи

1. Ознайомитися з наведеними вище теоретичними відомостями.
2. Виконати приклади, які приводяться в теоретичних відомостях.
3. Виконати наступні завдання:

3.1. Написати три окремі скрипти для кожної з частин завдання

3.2. На початку, у вигляді коментаря, буде містити назву курсу та номер лабораторної роботи , а також ваше ім'я та прізвище, та назву вашої групи

3.3. у першому рядку буде виводити назву курсу та номер лабораторної роботи (4.1, 4.2, 4.3)

3.4. у другому – буде виводити ваше ім'я та прізвище, а також номер Вашої заліковки.

Завдання 1

Використовуючи оператор циклу while, розв'язати наступні задачу (за варіантом):

1. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{(-1)^{n-1}}{n^n}$
2. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{1}{2^n} + \frac{1}{3^n}$
3. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{1}{((3n-2)(3n+1))}$
4. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{(2n-1)^3}{2^n}$
5. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{10^n}{n!}$

6. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{n!}{(2n)!}$
7. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{n!}{n^n}$
8. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{2^n n!}{n^n}$
9. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{3^n n!}{(3n)!}$
10. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{n!}{3n^n}$
11. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{(n!)^2}{2^{n^2}}$
12. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \lg(n!)e^{-n\sqrt{n}}$
13. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = 10^{-n}(n-1)!$
14. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{n^3}{(3n-3)!}$
15. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = \frac{n}{(n-1)^2}$
16. Знайти суму ряду з точністю ε=10-4, загальний член якого $a_n = e^n \cdot 100^{-n^2}$
17. Знайти суму 13 членів ряду, у якому $a_n = \frac{\ln(n!)}{n^2}$
18. Знайти суму 15 членів ряду, у якому $a_n = \frac{n^{\ln n}}{(\ln n)^n}$

Контрольні запитання.

1. Як описується та виконується оператор розгалуження?
2. Як описується та виконується оператор множинного розгалуження?
3. Що називається логічним виразом?
4. Які 3 можливих варіанти представлення умови в інструкції *if*?
5. Що таке цикл? Навіщо вони потрібні?

6. Як описується та виконується циклічна інструкція while?
7. Як можна організувати нескінченні цикли? Наведіть декілька варіантів і поясніть їх.