

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ

до самостійних робіт з дисципліни «Основи програмування на Python»

для бакалаврів спеціальності 123 "Комп'ютерна інженерія"
заочної форми навчання

2024

Методичні вказівки до самостійних робіт з дисципліни «Основи програмування на Python» для бакалаврів спеціальності 123 "Комп'ютерна інженерія", заочної форми навчання / Укл.: А.В. Тіменко, Н.А. Куликовська – Запоріжжя: НУ «Запорізька політехніка», 2024. – 30 с.

Укладач: А.В. Тіменко, старший викладач
Н.А. Куликовська, старший викладач

Рецензент: М.Ю. Тягунова, к.т.н., доцент

Відповідальний
за випуск: А.В. Тіменко, старший викладач

Затверджено:
На засіданні кафедри
«Комп'ютерні системи та мережі»
Протокол № 7
від 22 березня 2024р.

Рекомендовано до видання
НМК факультету КНТ
Протокол № 8
від 27 березня 2024 р.

ЗМІСТ

1 ЛАБОРАТОРНА РОБОТА №1 ОПАНУВАННЯ СТРУКТУРИ КОДУ	4
1.1 Теоретичні відомості	4
1.2 Хід роботи.....	7
1.3 Зміст звіту	8
1.4 Контрольні питання	8
2 ЛАБОРАТОРНА РОБОТА №2.....	10
ОПРАЦЮВАННЯ ФАЙЛІВ.....	10
2.1 Теоретичні відомості	10
2.2 Хід роботи.....	15
2.3 Зміст звіту	15
2.4 Контрольні запитання.....	16
3 ЛАБОРАТОРНА РОБОТА №3.....	17
ВИКОРИСТАННЯ ФУНКЦІЙ ДОСТУПУ ДО ОПЕРАЦІЙНОЇ СИСТЕМИ	17
3.1 Теоретичні відомості	17
3.2 Завдання до виконання.....	22
3.3 Зміст звіту	22
3.4 Контрольні запитання.....	22
ЛІТЕРАТУРА.....	24
ДОДАТОК А Стандартні функції Python.....	25
ДОДАТОК Б Методи	27

1 ЛАБОРАТОРНА РОБОТА №1 ОПАНУВАННЯ СТРУКТУРИ КОДУ

Мета роботи: отримати навички програмування на мові Python 3 й опанувати структури коду.

1.1 Теоретичні відомості

Програма Python складається з блоків коду. block — це частина тексту програми Python, яка виконується як одиниця. Це блоки: модуль, тіло функції та визначення класу. Кожна команда, введена в інтерактивному режимі, є блоком. Файл сценарію (файл, наданий інтерпретатору як стандартний ввід або вказаний як аргумент командного рядка для інтерпретатора) є блоком коду. Команда сценарію (команда, вказана в командному рядку інтерпретатора з опцією -c) — це блок коду. Модуль, запущений як сценарій верхнього рівня (як модуль `__main__`) з командного рядка за допомогою аргументу -m, також є блоком коду. Рядковий аргумент, який передається до вбудованих функцій `eval()` і `exec()`, є блоком коду.

Блок коду виконується у `execution frame`. Кадр містить деяку адміністративну інформацію (використовується для налагодження) і визначає, де і як виконання продовжується після завершення виконання блоку коду.

`Names` стосуються об'єктів. Імена вводяться за допомогою операцій зв'язування імен.

Ціль, що зустрічається в операторі `del`, також вважається зв'язаною для цієї мети (хоча фактична семантика полягає в тому, щоб роз'єднати ім'я).

Кожен оператор призначення або імпорту відбувається в блоці, визначеному визначенням класу чи функції, або на рівні модуля (блок коду верхнього рівня).

Якщо ім'я прив'язане до блоку, воно є локальною змінною цього блоку, якщо не оголошено як `nonlocal` або `global`. Якщо ім'я прив'язане на рівні модуля, воно є глобальною змінною. (Змінні блоку коду модуля є локальними та глобальними.) Якщо змінна використовується в блоці коду, але не визначена там, це `free variable`.

Кожне входження імені в текст програми посилається на binding цього імені, встановлене наступними правилами розпізнавання імен.

Scope визначає видимість імені в блоці. Якщо локальна змінна визначена в блоці, її область включає цей блок. Якщо визначення міститься у функціональному блоці, область поширюється на будь-які блоки, що містяться в визначальному, якщо тільки блок не вводить іншу прив'язку для імені.

Коли ім'я використовується в блоці коду, воно розпізнається за допомогою найближчої охоплюючої області. Набір усіх таких областей, видимих для блоку коду, називається environment блоку.

Якщо ім'я взагалі не знайдено, виникає виняток NameError. Якщо поточна область є областю функції, а ім'я посилається на локальну змінну, яка ще не прив'язана до значення в точці, де використовується ім'я, виникає виняток UnboundLocalError. UnboundLocalError є підкласом NameError.

Оператор global має ту саму область дії, що й операція прив'язки імені в тому самому блоці. Якщо найближча охоплююча область для вільної змінної містить глобальний оператор, вільна змінна розглядається як глобальна.

Простір імен для модуля створюється автоматично під час першого імпорту модуля. Основний модуль для сценарію завжди називається `__main__`.

```
class A:
    a = 42
    b = list(a + i for i in range(10))
```

Винятки — це засіб виходу зі звичайного потоку керування блоком коду для обробки помилок або інших виняткових умов. Виняток викликається в точці, де виявлено помилку; вона може оброблятися навколишнім блоком коду або будь-яким блоком коду, який прямо чи опосередковано викликав блок коду, де сталася помилка.

Інтерпретатор Python викликає виняток, коли виявляє помилку під час виконання (наприклад, ділення на нуль). Програма на Python також може явно викликати виняток за допомогою оператора raise. Обробники винятків визначаються оператором try ... except. Речення finally такого оператора можна використовувати для визначення коду

очищення, який не обробляє винятков, але виконується незалежно від того, чи виникла винятков у попередньому кодї.

Python використовує модель «припинення» обробки помилок: обробник винятків може дізнатися, що сталося, і продовжити виконання на зовнішньому рівні, але він не може виправити причину помилки та повторити невдалу операцію (за винятком повторного введення несправної частини коду зверху).

Якщо винятков взагалі не обробляється, інтерпретатор припиняє виконання програми або повертається до основного інтерактивного циклу. У будь-якому випадку він друкує зворотне трасування стека, за винятком випадків, коли винятком є SystemExit.

Винятки визначаються екземплярами класу. Речення `except` вибирається залежно від класу примірника: воно має посилатися на клас примірника або його невіртуальний базовий клас. Примірник може бути отриманий обробником і може містити додаткову інформацію про винятковий стан.

Алгоритми в Python – це набір інструкцій, написаних мовою програмування Python, які визначають порядок виконання операцій для розв’язання певного завдання. Алгоритми можуть бути різними за складністю та ефективністю, і вибір правильного алгоритму може суттєво вплинути на продуктивність програми та структуру коду.

Алгоритми, у яких використовується тільки структура «Слідування», називаються лінійними.

Алгоритми, в основі яких лежить структура «Розгалуження», називаються алгоритмами з розгалуженнями.

Алгоритми, в основі яких лежить структура «Повторення», називають циклічними.

На практиці алгоритми розв’язування складних задач містять у собі базові структури всіх трьох типів.

Лінійні алгоритми — це прості алгоритми, які виконуються послідовно, крок за кроком, без використання розгалужень або циклів. Вони розв’язують задачі, де послідовність виконання операцій є прямолінійною і не залежить від зовнішніх умов або змін даних.

Приклади задач, які часто розв’язуються за допомогою лінійних алгоритмів: обчислення простих математичних операцій, пошук максимального/мінімального елемента у списку, знаходження суми елементів списку та інші прості операції, які виконуються послідовно.

Для формування структури коду треба розглянути наступні етапи.

1. Визначити основні операції. Перед вибором структури даних, визначити, які операції будуть виконуватися. Наприклад, якщо потрібно часто виконувати пошук елементів за їхніми ключами, словник (dictionary) може бути відповідним вибором. Якщо необхідно зберігати впорядковані елементи з можливістю повторення, то список (list) або кортеж (tuple) може бути кращим.

2. Врахувати швидкість доступу. Деякі структури даних забезпечують швидкий доступ до елементів, наприклад, словники та множини. У той час як інші, такі як списки, можуть мати повільний доступ, особливо під час пошуку елементів.

3. Враховувати унікальність елементів. Якщо потрібно зберігати унікальні елементи, то множина (set) може бути підходящим вибором, оскільки вона автоматично видаляє дублікати. Якщо унікальність не є проблемою, тоді можна використовувати списки або інші структури.

4. Приділити увагу операціям вставки та видалення. Якщо завдання вимагає частих вставок і видалень елементів, то вибрати структуру даних, що забезпечує ефективні операції додавання та видалення елементів. Списки зазвичай забезпечують швидкі операції додавання та видалення в кінець, тоді як множини пропонують ефективні методи додавання та видалення елементів.

5. Розглянути обсяг даних. Якщо завдання включає великий обсяг даних, то звернути увагу на займану пам'ять і швидкість роботи структури даних. Деякі структури можуть споживати більше пам'яті, ніж інші, і забезпечувати повільні операції для великого обсягу даних.

6. Врахувати необхідність сортування. Якщо потрібно підтримувати елементи в упорядкованому вигляді, виберіть структуру даних, яка забезпечує ефективні методи сортування. Списки мають вбудовані методи сортування, тоді як словники та множини зазвичай не гарантують порядок елементів.

7. Врахувати, що вибір структури даних може залежати від конкретного контексту та специфіки завдання.

1.2Хід роботи

Напишіть програми у середовищі програмування для розв'язування таких завдань відповідно до варіанту:

1. Виведіть вітальне повідомлення для кожного користувача після його входу на сайт. Створіть список з кількох імен користувачів, включаючи ім'я 'Admin'. Перебираючи елементи списку, виведіть повідомлення для кожного користувача. Для користувача з ім'ям 'Admin' виведіть особливе повідомлення - наприклад: Hello Admin, I hope you're well.. У інших випадках виводиться універсальне привітання - наприклад: Hello Alex, thank you for logging in again.. Додайте команду if, яка перевірить, що список користувачів не порожній. Якщо список порожній, виведіть повідомлення: We need to find some users!. Видаліть зі списку всі імена користувачів і переконайтеся у тому, що програма виводить правильне повідомлення.

2. Потрібно визначити, чи є даний рік високосним. Нагадаємо, що високосними роками вважаються ті роки, порядковий номер яких або кратний 4, але при цьому не кратний 100, або кратний 400 (наприклад, 2000-й рік був високосним, а 2100-й буде невисокосним роком). Програма повинна коректно працювати на числах від 1900 до 3000 років. Виведіть Leap year. у разі, якщо рік є високосним і Ordinary year. у протилежному випадку.

1.3Зміст звіту

Звіт повинен мати такі складові:

- титульний лист;
- мета роботи;
- результат виконання завдань;
- відповіді на контрольні питання.

1.4Контрольні питання

1. Які два значення має булевий тип даних?
2. Назвіть три булевих оператори і шість операторів порівняння.
3. Напишіть кілька прикладів із таблиць істинності.
4. Яка різниця між операторами порівняння і присвоювання?
5. Що таке вказівка розгалуження і де вона використовується?
6. Наведіть приклади виразів, що можуть використовуватись як умова.

7. Яке сполучення клавіш використовується для виходу з нескінченного циклу?
8. Чим відрізняються виклики функцій `range(15)`, `range(0, 15)`, `range(0, 15, 1)` у циклі `for`?
9. Навіщо використовувати у програмах функції?
10. Коли виконується код функції: коли вона визначається, чи коли вона викликається?
11. З допомогою якої конструкції створюються функції?
12. Чим відрізняється визначення функції від її виклику?
13. Скільки глобальних областей видимості може мати програма на Python? Скільки локальних?
14. Що таке «значення, яке повертає функція»? Чи може бути це значення у вигляді виразу?
15. Що відбувається із змінними, які знаходяться у локальній області видимості, при поверненні їх із функції?
16. Яке значення повертає функція, якщо у ній відсутній оператор `return`?
17. Як змусити змінну у функції посилатися на глобальну змінну?
18. Що таке тип даних `None`?
19. Як можна запобігти аварійному завершенню програми?
20. Які блоки коду записують у блок `try`? А які у `except`?

2 ЛАБОРАТОРНА РОБОТА №2 ОПРАЦЮВАННЯ ФАЙЛІВ

Мета роботи: отримати навички програмування на мові Python з використанням файлів.

2.1 Теоретичні відомості

Перед тим як щось записати у файл або зчитати з нього, спочатку необхідно відкрити файл:

`fileobj = open(filename, mode)`

– `fileobj` - об'єкт файла, який повертається функцією **`open()`**;

– `filename` - рядок з назвою файла або шлях до файла;

– `mode` - рядок, який вказує на тип файла і дії, які можна виконувати над файлом;

Перша літера рядка `mode` вказує на **операцію з файлом**:

– `r` - означає **читання з файла**;

– `w` - означає **запис в існуючий файл** (якщо файла не існує, він буде створений);

– `x` - означає **запис у новий файл**, тобто, якщо файла не існує (він буде створений);

– `a` - означає **додавання в кінець файла**, якщо файл існує.

Друга літера рядка `mode` вказує на **тип файла**:

– `t` (або нічого) - означає, що **файл текстовий**;

– `b` - означає, що **файл бінарний**.

Після відкриття файла, виконують читання даних з файла або запис даних у файл.

При відкритті файла у текстовому режимі необхідно перетворити послідовності закінчення рядків у файлі, що специфічні для операційних систем (`\n` у Linux Ubuntu, `\r\n` у Windows, `\r` у MacOS), в `\n`.

Під час запису в текстовому режимі перетворення `\n` у закінчення рядків, які специфічні для платформи, відбувається за замовчуванням.

При відкритті файла можна вказати режим обробки символів нового рядка: записати вказівку `newline = 'value'`, де `value` може набувати значень `\n`, `\r`, `\r\n` або порожнього рядка, в результаті чого

тільки вказана послідовність буде використовуватися як ознака нового рядка:

```
fileobj = open(filename, mode, newline='\n')
```

Якщо файл відкривається у двійковому режимі, параметр `newline` не потрібен, тому що всі байти повертаються точно в тому вигляді, в якому вони зберігаються в файлі.

При читанні текстового файла Python повертає рядки у форматі Unicode. Python використовує кодування за замовчуванням. Щоб дізнатися про бажане кодування для вашої операційної системи, виконайте наступний код:

```
import locale
print(locale.getpreferredencoding())
Результат отриманий для Windows:
cp1251
```

Наприклад, при читанні текстового файла в системі з кодуванням UTF-8, Python намагатиметься декодувати інформацію з файла у UTF-8. Якщо інформація у файлі кодувалася з UTF-8, все пройде успішно. Якщо ж файл кодувався у іншому кодуванні, ви отримаєте помилку або незрозумілий текст.

Щоб уникнути цього, необхідно явно передати назву кодування з параметром `encoding` функції **open()**, наприклад:

```
data = open('filename.txt', 'rt', encoding='cp1251').read()
```

При записі в текстовий файл можна також явно задати параметр кодування.

Потім необхідно **файл закрити** функцією **close()**:

```
fileobj.close()
```

8.2. Запис даних у текстові файли

Використаємо як джерело даних фрагмент вірша Тараса Шевченка англійською мовою, що міститься у змінній `poem` і запишемо цей вірш у текстовий файл **result.txt**:

```
poem = """The mighty Dnieper roars and bellows,
The wind in anger howls and raves,
Down to the ground it bends the willows,
And mountain-high lifts up the waves."""
frecord = open('result.txt', 'wt')
a = frecord.write(poem)
frecord.close()
print(a)
```

Функція **write()**, значення якої було присвоєне змінній *a*, повертає число записних у файл байтів

```
151
```

але не додає ніяких інших символів у файл, як це робить функція **print()**, яка також дозволяє записувати у файл:

```
frecord = open('result.txt', 'wt')
print(poem, file=frecord)
frecord.close()
```

Для запису у файл можна використовувати обидві функції. Варто лише пам'ятати про два аргументи функції **print()**:

- *sep* - розділювач (за замовчуванням це **пропуск ' '**);
- *end* - символ кінця файла (за замовчуванням це **символ нового рядка \n**).

Для того щоб функція **print()** працювала так само, як функція **write()**, змінений код виглядатиме так:

```
frecord = open('result.txt', 'wt')
print(poem, file=frecord, sep=", end=")
frecord.close()
```

Вміст файла **result.txt** буде таким:

```
The mighty Dnieper roars and bellows,
The wind in anger howls and raves,
Down to the ground it bends the willows,
And mountain-high lifts up the waves.
```

Перевіримо, чи «**врятує**» режим *x* файл **result.txt**

```
frecord = open('result.txt', 'xt')
```

від його перезаписування?

```
Traceback (most recent call last):
```

```
File "C:\Python34\test.py", line 1007, in <module>
```

```
    frecord = open('result.txt', 'xt')
```

```
FileExistsError: [Errno 17] File exists: 'result.txt'
```

Перезаписати не вдалося, оскільки у режимі *x* запис у файл здійснюється лише у новий файл. Виникла помилка і згенерувався виняток. У такому випадку корисно використовувати обробник винятку:

```
try:
    frecord = open('result.txt', 'xt')
    frecord.write(poem)
except FileExistsError:
```

```
print('result.txt already exists!')
```

Результатом використання обробника винятку буде повідомлення про те, що файл з таким ім'ям вже існує і записати у нього інформацію неможливо:

```
result.txt already exists!
```

Щоб прочитати увесь вміст файла за один раз, треба використовувати функцію **read()**.

```
freading = open('result.txt', 'rt' )
```

```
poem = freading.read()
```

```
freading.close()
```

```
print(poem)
```

```
print(len(poem))
```

Результат читування даних з файла буде наступним:

```
The mighty Dnieper roars and bellows,
```

```
The wind in anger howls and raves,
```

```
Down to the ground it bends the willows,
```

```
And mountain-high lifts up the waves.
```

```
151
```

Можна вказати максимальну кількість символів, які зчитуються з файла за один раз. Для прикладу, будемо зчитувати по 60 символів за раз і приєднувати кожний фрагмент до рядка poem, щоб відтворити увесь вміст файла:

```
poem = "
```

```
freading = open('result.txt', 'rt' )
```

```
chunk = 60
```

```
while True:
```

```
    fragment = freading.read(chunk)
```

```
    if not fragment:
```

```
        break
```

```
    poem += fragment
```

```
    print(len(fragment))
```

```
freading.close()
```

Після того як зчитали увесь файл, подальші виклики функції **read()** будуть повертати порожній рядок ("). Це буде оцінено як False в перевірці if not fragment і дозволить вийти з нескінченного циклу while True.

Можна також зчитувати файл по одному рядку за раз за допомогою функції **readline()**. У наступному прикладі будемо

приєднувати кожен рядок до рядка poem, щоб відтворити увесь вміст файла:

```
poem = ""
freading = open('result.txt', 'rt')
while True:
    line = freading.readline()
    if not line:
        break
    poem += line
freading.close()
print(poem)
```

Коли весь файл буде зчитаний, функція **readline()** (як і функція **read()**) поверне порожній рядок, що буде вважатися False і відбудеться вихід з циклу.

Найпростіший спосіб прочитати текстовий файл - використовувати цикл. Він буде повертати по одному рядку за раз. Цей приклад схожий на попередній, але коду у ньому менше:

```
poem = ""
freading = open('result.txt', 'rt')
for line in freading:
    poem += line
freading.close()
print(poem)
```

А функція **readlines()** повертає список зчитаних рядків:

```
freading = open('result.txt', 'rt')
lines = freading.readlines()
freading.close()
print(len(lines), 'lines read')
print(lines)
for line in lines:
    print(line.rstrip())
```

Функція **rstrip()** видаляє символи нового рядка `\n` у кожному із виведених рядків, окрім останнього, в якому цей символ відсутній.

```
4 lines read
```

```
['The mighty Dnieper roars and bellows,\n', 'The wind in anger howls
and raves,\n', 'Down to the ground it bends the willows,\n', 'And mountain-
high lifts up the waves.']
```

```
The mighty Dnieper roars and bellows,
```

The wind in anger howls and raves,
Down to the ground it bends the willows,
And mountain-high lifts up the waves.

У Python присутні **менеджери контексту**, що використовуються для очищення об'єктів (звільнення ресурсів), на зразок відкритих файлів. Для роботи з файлами менеджер контексту має таку конструкцію:

with вираз as змінна:

блок коду

Наприклад:

```
with open('result.txt', 'wt') as freading:
    freading.write(poem)
```

Після того як блок коду, розташований у менеджері контексту (у цьому випадку один рядок `freading.write(poem)`), завершиться (або звичайним способом, або шляхом генерації винятку), файл буде закритий автоматично.

2.2 Хід роботи

Напишіть програми у середовищі програмування для розв'язування таких завдань відповідно варіанту:

1. Створіть новий файл у текстовому редакторі і напишіть кілька рядків тексту у ньому про можливості Python. Кожен рядок повинен починатися з фрази: `In Python you can ....` Збережіть файл з ім'ям **learning_python.txt**. Напишіть програму, яка зчитує файл і виводить текст з перебором рядків об'єкта файла і зі збереженням рядків у списку з подальшим виведенням списку поза блоком **with**.

2. Створіть порожній файл **guest_book.txt** у текстовому редакторі. Напишіть цикл **while**, який запитує у користувачів імена. При введенні кожного імені виведіть на екран рядок з вітанням для користувача і запишіть рядок вітання у файл з ім'ям **guest_book.txt**. Простежте за тим, щоб кожне повідомлення розміщувалося в окремому рядку файла. Передбачте вихід з циклу.

2.3 Зміст звіту

Звіт повинен мати такі складові:

— титульний лист;

- мета роботи;
- результат виконання завдань;
- відповіді на контрольні питання.

2.4 Контрольні запитання

1. Які аргументи задають режим відкриття файла з допомогою функції `open()`?
2. За допомогою яких функцій можна записувати у файл?
3. Чим відрізняється робота функцій `read()`, `readline()` і `readlines()`?
4. Для чого використовують у роботі з файлами оператор `with`?
5. Як створити і прочитати бінарний файл?
6. Які засоби Python використовуються для відслідковування місцезнаходження конкретних байтів у бінарних файлах?
7. Як використовуються функції `csv.reader()` і `csv.writer()`?
8. Яка функція приймає JSON-рядок і повертає структуру даних Python?
9. Яка функція приймає структуру даних Python, а повертає рядок JSON-даних?
10. Які інструменти Python використовують для отримання доступу до баз даних і роботи з ними?
11. Що таке SQL?
12. Наведіть приклад запиту з використанням мови SQL.

3 ЛАБОРАТОРНА РОБОТА №3 ВИКОРИСТАННЯ ФУНКЦІЙ ДОСТУПУ ДО ОПЕРАЦІЙНОЇ СИСТЕМИ

Мета роботи: отримати навички програмування на мові Python з використанням доступу до операційної системи.

3.1 Теоретичні відомості

Робота з файловою системою передбачає створення, перейменування, переміщення або звернення до файлів і каталогів. Усі ці та багато інших завдань можна виконати за допомогою програм, написаних на Python.

Python надає багато системних функцій, що містяться в модулі **os** (скорочення від Operating System - операційна система), який необхідно імпортувати для роботи програм.

Частина назви файла, яка слідує за останньою крапкою, називається **розширенням файла** і вказує на **тип файла**. Розширення файла, зазвичай, надає програма, у якій він був створений.

Наприклад, файл з назвою **projects.docs** має тип файла - документ Word, про що свідчить розширення .docs. Частина назви файла (projects) до останньої крапки - це **ім'я файла**, яке вказує користувач.

Файл має дві ключові ознаки: **назва файла** і **шлях до файла**.

Шлях визначає, де саме на комп'ютері знаходиться файл. Наприклад, файл з назвою **projects.docs** може мати шлях до файла C:\Users\gt\Documents. Це означає, що файл **projects.docs** зберігається у каталозі **Documents**.

Частина C:\ шляху до файла - це **кореневий каталог**, який містить усі інші каталоги (**gt, Users**) і сам файл **projects.docs**.

У системах Windows кореневим каталогом є C:\, а шляхи до файлів записують з використанням **зворотного слешу (\)**, який розділяє імена каталогів. В Linux Ubuntu кореневим каталогом є каталог /, а розділення імен каталогів виконують з використанням **прямого слеша (/)**.

Шлях до файлу (або каталогу), який враховує використання обох слешів, можна сформувати з використанням функції **join()**, яка міститься у модулі **path**, який у свою чергу міститься у модулі **os**.

Якщо передати у цю функцію значення назв файлів і каталогів й виконати наступний код

```
import os
print(os.path.join('usr', 'bin', 'media'))
```

функція поверне значення, у якому сформується шлях з коректною підстановкою слеша-розділювача:

```
usr\bin\media
```

Якщо ж виконати даний приклад в інтерактивному режимі інтерпретатора Python, виклик функції **os.path.join()** поверне рядок:

```
'usr\bin\media'
```

Якщо викликати вищезгадану функцію **os.path.join()** в Linux Ubuntu, вона поверне рядок:

```
'usr/bin/media'
```

Отже, для формування шляхів до файлів або каталогів у вигляді рядків використовується функція **os.path.join()**. Наприклад, наступний код

```
import os
myfiles = ['accounts.txt', 'details.csv', 'invite.docs']
for filename in myfiles:
    print(os.path.join('C:\\Users\\gt\\Documents', filename))
```

формує шляхи до файлів, додаючи до імен каталогів назви файлів зі списку:

```
C:\Users\gt\Documents\accounts.txt
```

```
C:\Users\gt\Documents\details.csv
```

```
C:\Users\gt\Documents\invite.docs
```

Кожна програма, яка виконується на комп'ютері, має свій **поточний робочий каталог** (current working directory - cwd).

Будь-які шляхи до файлів або каталогів, які не починаються з кореневого каталогу, задаються відносно поточного робочого каталогу.

Для отримання значення поточного робочого каталогу у вигляді рядка використовується функція **os.getcwd()**, а для переходу з поточного робочого каталогу у інший каталог - функція **os.chdir()**.

```
import os
print(os.getcwd())
os.chdir('C:\\Windows\\Temp')
```

```
print(os.getcwd())
```

У даному випадку, в якості робочого поточного каталогу встановлюється каталог по шляху C:\Python36, відбувається перехід у інший каталог, який стає поточним:

```
C:\Python36
```

```
C:\Windows\Temp
```

Існує два способи визначення шляху до файлу або каталогу:

- **абсолютний шлях** - завжди починається з імені кореневого каталогу;

- **відносний шлях** - задається відносно поточного робочого каталогу програми.

Існують також каталоги, які позначаються одним (.) або двома (..) символами крапки. Це не реальні каталоги, а спеціальні імена, які використовуються для визначення шляхів.

Для створення нових каталогів використовується функція **os.makedirs()**, яка створює усі необхідні проміжні каталоги:

```
os.makedirs('C:\\Python36\\other\\myprograms')
```

Модуль **os.path** містить функції для визначення абсолютного шляху по заданому відносному, а також для перевірки того, чи шлях є абсолютним.

Виклик **os.path.abspath(path)** - повертає рядок абсолютного шляху до файлу або каталогу.

Виклик **os.path.isabs(path)** - повертає True, якщо шлях є абсолютним, і False у протилежному випадку, коли шлях виявиться відносним.

Виклик **os.path.relpath(path, start)** - повертає рядок відносного шляху від точки start до path. Якщо start не вказаний, то в якості нього використовується поточний робочий каталог.

На наступному прикладі коду:

```
import os
```

```
print(os.path.abspath('.'))
```

```
print(os.path.abspath('.\\Scripts'))
```

```
print(os.path.isabs('.'))
```

```
print(os.path.isabs(os.path.abspath('.')))
```

```
print(os.path.relpath('C:\\Windows', 'C:\\'))
```

```
print(os.path.relpath('C:\\Windows', 'C:\\Python36\\Scripts'))
```

```
print(os.getcwd())
```

робота цих функцій виглядає так:

```

C:\Python36
C:\Python36\Scripts
False
True
Windows
..\..\Windows
C:\Python36

```

Для отримання із повного шляху лише назви файла використовують функцію **os.path.basename()**. Якщо необхідно отримати частину шляху без назви файла використовують функцію **os.path.dirname()**.

Якщо шлях до файла вважати таким рядком 'C:\\Windows\\System32\\calc.exe', то ці дві функції:

```

import os
path = 'C:\\Windows\\System32\\calc.exe'
print(os.path.basename(path))
print(os.path.dirname(path))

```

відпрацюють так:

```

calc.exe
C:\\Windows\\System32

```

Щоб отримати одночасно і назву файла, так і імена каталогів на шляху до нього, можна скористатися функцією **split()** у такому виконанні:

```

import os
path = 'C:\\Windows\\System32\\calc.exe'
print(os.path.split(path))

```

Результатом виведення буде кортеж з двох елементів:

```

('C:\\Windows\\System32', 'calc.exe')

```

Щоб отримати список усіх каталогів на шляху до файла і назву самого файла використовують розділювач **os.path.sep**.

```

import os
path = 'C:\\Windows\\System32\\calc.exe'
print(path.split(os.path.sep))

```

Для системи Windows буде такий результат:

```
['C:', 'Windows', 'System32', 'calc.exe']
```

Для системи Linux Ubuntu

```

import os
print('/usr/bin'.split(os.path.sep))

```

першим елементом такого списку буде порожній рядок:

```
['', 'usr', 'bin']
```

У модулі **os.path** присутні функції, які дозволяють обчислювати розміри файлів у байтах і визначати, які файли та каталоги знаходяться у вказаному каталозі. До таких функцій належать:

- **os.path.getsize(path)** - повертає у байтах розмір файла, шлях до якого вказаний у path;

- **os.listdir(path)** - повертає список рядків з назвами усіх файлів, шлях до яких вказаний у path.

Коли виконати поданий код

```
import os
print(os.path.getsize('C:\\Windows\\System32\\calc.exe'))
print(os.listdir('C:\\Windows\\System32'))
```

отримаємо розмір у байтах вказаного файла і список усіх файлів вказаного каталогу.

Для знаходження сумарного розміру файлів, наприклад, у каталозі по шляху C:\\Python36, можна використати таку програму:

```
total = 0
for filename in os.listdir('C:\\Python36'):
    total = total + os.path.getsize(os.path.join('C:\\Python36',
filename))
print(total)
```

Результатом виконання даної програми буде розмір у байтах усіх файлів каталогу по шляху C:\\Python36:

```
4582106
```

Функції Python для роботи з файлами і каталогами можуть повідомляти про помилки, коли не знаходять шлях до файла або каталогу. Модуль **os.path** містить функції, які перевіряють, чи існує шлях до каталогу або файла:

- **os.path.exists(path)** - повертає значення True, якщо **файл (або каталог)** за вказаним шляхом path існує, і значення False у протилежному випадку;

- **os.path.isfile(path)** - повертає значення True, якщо **файл** за вказаним шляхом path існує, і значення False у протилежному випадку;

- **os.path.isdir(path)** - повертає значення True, якщо **каталог** за вказаним шляхом path існує, і значення False у протилежному випадку.

3.2 Завдання до виконання

Напишіть програми у середовищі програмування для розв'язування таких завдань:

1. Запишіть програмно поточні дату й час як рядок у текстовий файл `today.txt`. Зчитайте текстовий файл `today.txt` і розмістіть дані у рядку `today_string`. Розберіть дату з рядка `today_string` на складові, використовуючи функції для роботи з датою й часом, і виведіть їх на екран у вигляді повідомлень з пояснюючим текстом.

2. Створіть об'єкт `harry`, що містить дату вашого народження. Використайте функції для роботи з датою і часом, щоб дізнатися відповіді на такі питання (виведення організуйте в окремий файл у вигляді, на зразок «запитання: відповідь», у окремих рядках):

- а) "Яка дата вашого народження?"
- б) "У який день тижня ви народилися?"
- с) "Коли вам буде (або вже було) 13 330 днів від народження?"

3.3 Зміст звіту

Звіт повинен мати такі складові:

- титульний лист;
- мета роботи;
- результат виконання завдань;
- відповіді на контрольні питання.

3.4 Контрольні запитання

1. Відносно чого задається відносний шлях?
2. З чого починається абсолютний шлях?
3. Яке призначення функцій `os.getcwd()` і `os.chdir()`?
4. Що позначають імена `.` і `..`?
5. Які частини шляху `C:\Users\gt\logging\log12042019.txt` представляють імена каталогів та ім'я файлу?
6. Які функції використовуються для копіювання, перейменування і видалення файлів і каталогів?
7. Що таке «епоха Unix»?

8. Яка функція повертає кількість секунд, які пройшли з моменту початку «епохи Unix»?
9. У чому різниця між об'єктами `datetime` і `timedelta`?
10. Які інструменти локалізації присутні у Python?

JITEPATYPA

1. Barry P. Head First Python: A Brain-Friendly Guide / Paul Barry. – [S. l.] : O'Reilly Media, Incorporated, 2023.
2. Borjigin C. Python Data Science / Chaolemen Borjigin. – [S. l.] : Springer, 2023.
3. Chongchitnan S. Exploring University Mathematics with Python [Electronic resource] / Siri Chongchitnan. – Cham : Springer Nature Switzerland, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-46270-2>
4. Deep Learning and Its Applications Using Python / Niha K. Basha [et al.]. – [S. l.] : Wiley & Sons, Incorporated, John, 2023.
5. Hazrat R. A Course in Python [Electronic resource] / Roozbeh Hazrat. – Cham : Springer Nature Switzerland, 2023. – Mode of access: <https://doi.org/10.1007/978-3-031-49780-3>
6. Korites B. Python Graphics [Electronic resource] / Bernard Korites. – Berkeley, CA : Apress, 2023. – Mode of access: <https://doi.org/10.1007/978-1-4842-9660-8>
7. Kouh M. Thermal Physics Tutorials with Python Simulations / Minjoon Kouh, Taejoon Kouh. – [S. l.] : Taylor & Francis Group, 2023.
8. Matuszek D. Quick Python 3 / David Matuszek. – [S. l.] : CRC Press LLC, 2023.
9. Miller R. Computer Scripting Techniques with Python / Rick Miller. – [S. l.] : Pulp Free Press, 2023.
10. Nelli F. Python Data Analytics [Electronic resource] / Fabio Nelli. – Berkeley, CA : Apress, 2023. – Mode of access: <https://doi.org/10.1007/978-1-4842-9532->
11. Python Programming Recipes for IoT Applications / Madhusudan Ganuji Lanjewar [et al.]. – [S. l.] : Springer, 2023.

ДОДАТОК А

Стандартні функції Python

Виклик функції	Опис
help(x)	Документація об'єкта x
dir(obj)	Список імен/атрибутів об'єкта obj в алфавітному порядку
type(x)	Тип об'єкта x
id(x)	Ідентифікатор (адреса в пам'яті) об'єкта x
print(x)	Виведення x на екран або запис у текстовий файл
open(file, mode)	Відкриття файлу file в одному із режимів mode і повертання відповідного файлового об'єкта
input(x)	Зчитування і повертання рядка x вхідних даних
x.format()	Відформатоване подання рядка x, яке контролюється специфікатором формату
hash(x)	Значення хеш-функції об'єкта x
int(x)	Перетворення об'єкта x в ціле число
float(x)	Перетворення об'єкта x в дійсне (дробове) число
str(x)	Перетворення об'єкта x в рядок
chr(x)	Повертає символ (у вигляді рядка), чия позиція коду для Юнікод дорівнює зазначеному цілому x. Аргумент повинен розташовуватися в межах від 0 до 1,114,111 (0x10FFFF), в іншому випадку порушується виняток ValueError
ord(x)	Повертає для зазначеного Юнікод-символу ціле число, що представляє його позицію коду
round(x)	Найближче ціле число до числа x
abs(x)	Абсолютне значення x
pow(a, b)	Піднесення числа a до степеня b
min(x)	Мінімальний елемент в ітерованому x
max(x)	Максимальний елемент в ітерованому x
sum(x)	Сума елементів в ітерованому x
len(x)	Довжина ітерованого x (кількість елементів в x)
sorted(x)	Новий ітерований об'єкт, що містить елементи x у відсортованому порядку

reversed(x)	Новий ітерований об'єкт, що містить елементи x в зворотному порядку
tuple(x)	Новий кортеж tuple, створений з елементів x
list(x)	Новий список list, створений з елементів x
set(x)	Нова множина set, створена із елементів x
range(i, j)	Новий ітерований об'єкт, який містить числа від i до j-1 (стандартне значення i - 0)

ДОДАТОК Б

Методи

Таблиця Б.1 - Методи рядків

Метод	Призначення
s.capitalize()	Повертає копію рядка s, роблячи першу букву у верхньому регістрі
s.lower()	Повертає копію рядка s із символами у нижньому регістрі
s.swapcase()	Повертає копію рядка s, в якій кожна буква матиме протилежний регістр
s.title()	Повертає копію рядка s, в якій кожне нове слово починається з великої літери
s.upper()	Повертає копію рядка s із символами у верхньому регістрі
s.count(x)	Для рядка s повертає кількість входжень в нього зазначеного підрядка x
s.find(x)	Повертає найменший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок x не знайдено, повертає -1)
s.index(x)	Повертає найменший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок не знайдено, створюється виняток ValueError)
s.rfind(x)	Повертає найбільший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок x не знайдено, повертає -1)
s.rindex(x)	Повертає найбільший індекс з рядка s, за яким знаходиться початок зазначеного підрядка x (якщо підрядок не знайдено, створюється виняток ValueError)
s.replace(a, b)	Повертає копію рядка s, де всі входження підрядка a у рядок s замінюються підрядком b
s.startswith(x)	Повертає True, якщо рядок s починається з вказаного префіксу x (якщо ні, то повертає False)
s.endswith(x)	Повертає True, якщо рядок s закінчується вказаним префіксом x (якщо ні, то повертає False)

s.join(x)	Повертає рядок, складений з елементів ітеративного об'єкта x з розділювачами s
s.lstrip(chars)	повертає копію рядка s, на початку якого видалені символи chars
s.rstrip(chars)	повертає копію рядка s, наприкінці якого видалені символи chars
s.strip(chars)	повертає копію рядка s, на початку і у кінці якого видалені символи chars
s.split(char)	Розділення рядка s по розділювачу char і зберігання у список
s.center(width, fill)	Повертає відцентрований рядок, по краях якого стоїть символ fill (пропуск за замовчуванням)
s.ljust(width, fillchar)	Робить довжину рядка s не меншою width, в разі потреби заповнюючи останні символи символом fillchar
s.rjust(width, fillchar)	Робить довжину рядка s не меншою width, в разі потреби заповнюючи перші символи символом fillchar
s.isalnum()	Чи складається рядок s з букв і цифр?
s.isalpha()	Чи складається рядок s з букв?
s.isdigit()	Чи складається рядок s з цифр?
s.istitle()	Чи починаються слова в рядку s з великої букви?
s.isupper()	Чи складається рядок s із символів у верхньому регістрі?
s.islower()	Чи складається рядок s із символів у нижньому регістрі?

Таблиця Б.2 - Методи списків

Метод	Призначення
list.append(x)	Додає елемент x в кінець списку list
list.extend(L)	Розширює список list, додаючи у кінець всі елементи списку L
list.insert(i, x)	Вставляє у списку list на i-ий елемент значення x
list.remove(x)	Видаляє перший елемент у списку list, який має значення x; ValueError, якщо такого елемента не існує

list.pop(i)	Видаляє із списку list i-ий елемент і повертає його; якщо індекс не вказано, видаляється останній елемент
list.index(x, start, end)	Повертає положення у списку list першого елемента зі значенням x (при цьому пошук ведеться від start до end)
list.count(x)	Повертає кількість елементів із списку list зі значенням x
list.sort(reverse=False)	Сортує елементи списку list; якщо необов'язковий параметр reverse має значення True, список сортується в зворотному порядку
list.reverse()	Розвертає список list
list.copy()	Поверхнева копія списку list
list.clear()	Очищає список list

Таблиця Б.3 - Методи словників

Метод	Призначення
dict.clear()	Очистити словник dict
dict.copy()	Повертає копію словника dict
dict.fromkeys(seq, value=None)	Створює словник dict з ключами з послідовності seq і значенням value (за замовчуванням None)
dict.get(key, default=None)	Повертає значення ключа із словника dict, але якщо ключ відсутній, не створює виняток, а повертає default (за замовчуванням None)
dict.items()	Повертає пари (ключ, значення) із словника dict у вигляді списку кортежів [(key1, value1), (key2, value2), ...]
dict.keys()	Повертає ключі словника dict
dict.pop(key, default)	Видаляє ключ і повертає значення словника dict; якщо ключ відсутній, повертає default (якщо default відсутній - створює виняток KeyError)
dict.popitem()	Видаляє і повертає пару (ключ, значення) словника dict; якщо словник порожній, створює виняток KeyError (пам'ятайте, що словники

	невпорядковані)
dict.setdefault(key, default=None)	Повертає значення ключа словника dict, але якщо він відсутній, не створює виняток, а створює ключ із значенням default (за замовчуванням None)
dict.update(other)	Оновлює словник dict, додаючи пари (ключ, значення) із other; існуючі ключі перезаписуються (повертає None)
dict.values()	Повертає значення із словника dict у вигляді списку [value1, value2, ...]