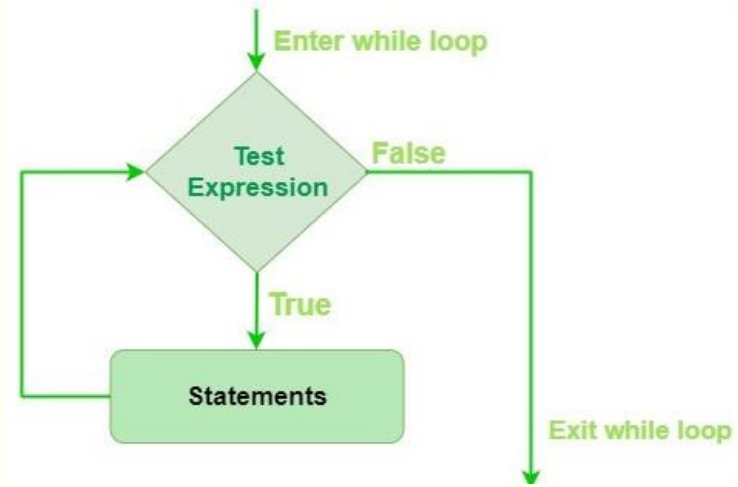
УПРАВЛЯЮЧІ ОПЕРАТОРИ МОВИ ПРОГРАМУВАННЯ PYTHON

Питання 6.3.

Оператор циклу while



Синтаксис оператора while:

```
while expression:
    statement(s)
```

- Управляючий вираз зазвичай включає одну або більше змінних, які ініціалізуються до початку циклу (або за допомогою моржового оператора), а потім змінюються десь у тілі циклу.
- Інструкції (statement(s)) представляють блок коду, який буде повторно виконуватись. Його часто називають тілом циклу та позначають відступами.
- Коли цикл while виконується, спочатку знаходиться значення управляючого виразу в контексті Boolean. Якщо його значення True, тіло циклу виконується.
- Потім управляючий вираз перевіряється знову, а цикл виконується, поки вираз не набуває значення False.

```
count = 0
while (count < 3):
    count = count + 1
    print("Hello Geek")

print()

# перевіряє, чи список ще має елементи
a = [1, 2, 3, 4]
while a:
    print(a.pop())
```

```
loops ×
D:\PycharmProjects\BuiltinDataSt
Hello Geek
Hello Geek
Hello Geek

4
3
2
1
```

Оператор циклу while

- Конструкції do-while у Python немає.
  - Зімітувати її можна кількома способами:

```
while True:
  stuff()
  if fail_condition:
    break
```

  - або так:

```
stuff()
while not fail_condition:
  stuff()
```

  - або так:

```
check = True
while check:
   do_stuff()
   check=condition()
```

Оператор циклу for

- Найбільш базова форма циклу for – обхід простого числового діапазону від початкового до кінцевого значення.
  - Може використовуватись функція range().
  - У прикладі діапазон чисел – [0; 5) з кроком 2:

```
for n in range(0, 5, 2):
    print(n)
```

```
0
2
4
```

```
for n in range(5, 0, -1):
    print(n)
```

```
5
4
3
2
1
```

  - С-подібний синтаксис циклу for (з трьох частин) у Python не підтримується.

- Цикл з ітератором (Collection-Based Loop, Iterator-Based Loop) проходить (ітерує) по колекції об’єктів, а не заданим числовим значенням чи умовам.

```
str = 'TEXT'
for ch in str:
    print(ch)
```

```
T
E
X
T
```

```
list = ['T', 'E', 'X', 'T']
for elem in list:
    print(elem)
```

```
T
E
X
T
```

Оператор галуження (if statement)

```
import math

if radius >= 0:
    print("Довжина кола = ", 2 * math.pi * radius)
    print("Площа = ", math.pi * radius ** 2)
else:
    print("Будь ласка, введіть додатнє число")
```

```
Введіть радіус: 3
Довжина кола =  18.84955592153876
Площа =  28.274333882308138
```

```
Введіть радіус: -3
Будь ласка, введіть додатнє число
```

```
password = input("Введіть пароль: ")
if password == "sshh":
    print("Ласкаво просимо!")
else:
    print("Доступ заборонено!")
```

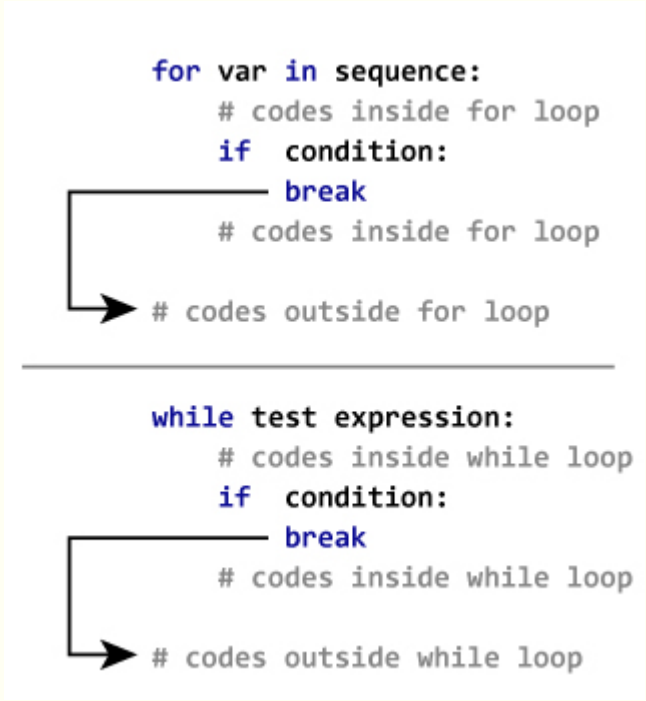
```
Введіть пароль: ddeee
Доступ заборонено!
```

```
score = int(input("Введіть вашу оцінку: "))
if score >= 90:
    print("Відмінно! Ваша оцінка A")
elif score >= 80:
    print("Дуже добре! Ваша оцінка - B")
elif score >= 70:
    print("Добре! Ваша оцінка - C")
elif score >= 60:
    print("Ваша оцінка - D. Варто повторити матеріал.")
else:
    print("Ви не здали екзамен!")
```

```
Введіть вашу оцінку: 76
Добре! Ваша оцінка - C
```

- Чи є в Python оператор switch?

Оператори безумовного переходу break, continue,



- Приклад використання оператору continue:

```
i = 0
a = 'geeksforgeeks'


while i < len(a):
    if a[i] == 'e' or a[i] == 's':
        i += 1
        continue
    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : g
Current Letter : k
Current Letter : f
Current Letter : o
Current Letter : r
Current Letter : g
Current Letter : k
```

- Приклад використання оператору break:

```
i = 0
a = 'geeksforgeeks'


while i < len(a):
    if a[i] == 'e' or a[i] == 's':
        i += 1
        break
    print('Current Letter :', a[i])
    i += 1
```

```
Current Letter : g
```

Блок else у циклах while і for

```
for f in files:
    if f.uuid == match_uuid:
        break
else:
    raise FileNotFound()
```

- Блок else знаходиться на одному рівні з блоком for, тому здається, що це деяка умова, яка стосується списка files.
  - Наприклад, коли for не знайшов жодного запису.
- Блок else виконується лише тоді, коли умова циклу стає False.
  - Блок else після циклів відноситься не до самого циклу, а до оператора break!
  - Якщо цикл переривається (break, return або raise) чи в ньому викидається виняток, блок else виконуватись не буде.
  - Читати приклад слід так: «якщо файл має потрібний uuid, то закінчити цикл; інакше викинути виняток».

Оператор перехоплення винятків try-except

- Використання try-except є найбільш розповсюдженим та природнім способом обробки непередбачених помилок.
  - Обробка помилок або винятків у Python може здійснюватись шляхом налаштування винятків.
  - Використовуючи блок try, можна реалізувати виняток та обробити помилку всередині блоку винятків.
  - При кожному перериванні коду всередині блоку try звичайний потік коду зупиняється, а елемент управління перемикається на блок except для обробки помилки.
- ***Навіщо використовувати конструкції Try-Except / Try-Except-else?***
  - Ви можете уникнути багатьох невідомих проблем, які можуть виникнути в коді.
  - Інколи код критично залежить від певної інформації, яка може застаріти до моменту її отримання.
  - Наприклад, код, що викликає os.path.exists чи Queue.full, може завершитись помилкою, оскільки ці функції можуть повертати дані, які застарівають до того часу, коли ви їх використовуєте.
  - Більш розумний вибір: дотримуватись стилю try-except-else, щоб надійніше керувати такими випадками.

Оператор перехоплення винятків try-except

- ***Викидання винятків також допустиме в Python.***
  - Просто викликається raise Exception(‘Test error!’).
  - Викинутий вняток зупинить поточне виконання та піде далі в стек викликів, поки не буде обробленим.
- ***Навіщо використовувати винятки?***
  - Вони не лише допомагають вирішувати популярні проблеми, але й корисні для контролю помилок у таких областях, як цикли, обробка файлів, обмін даними з БД, доступ до мережі тощо.
- Іноді потрібно мати можливість обробити довільний виняток, а також відобразити повідомлення про помилку чи виняток.
  - Під час тестування можна помістити код всередину блоку try:

```
try:
    # ваш код
except Exception as ex:
    print(ex)
```

  - Можна спіймати кілька винятків в одному except-блоці:

```
except (Exception1, Exception2) as e:
    pass
```

```
try:
    file = open('input-file', 'open mode')
except (IOError, EOFError) as e:
    print("Testing multiple exceptions. {}".format(e.args[-1]))
```

Оператор перехоплення винятків try-except

- Ви можете додати довільну кількість Except-блоків:

```
try:
    file = open('input-file', 'open mode')
except EOFError as ex:
    print("Caught the EOF error.")
    raise ex
except IOError as ex:
    print("Caught the I/O error.")
    raise ex
```

- Винятки, які виникли, продовжують переміщуватись до викликаючих методів, поки не будуть оброблені.
  - Також можна додати виняткову інструкцію - raise без аргументів.
  - Це призведе до перегляду винятку:

```
106  try:
107      # Навмисно викидаємо виняток
108      raise Exception('I learn Python!')
109  except:
110      print("Entered in except.")
111      # Відновлюємо виняток
112      raise
```

```
Traceback (most recent call last):
  File "D:/PycharmProjects/BuiltinDataStructures/loops.py", line 108, in <module>
    raise Exception('I learn Python!')
Exception: I learn Python!
Entered in except.
```

Оператор перехоплення винятків try-except. Застосування else

```
while True:
    x = int(input())

    try:
        result = 1 / x
    except:
        print("Error case")
        exit(0)
    else:
        print("Pass case")
        exit(1)
```

- Використовуйте блок else відразу пілся блоку try-except.
  - Блок else буде виконано тільки якщо не було згенеровано виняток.
  - Оператору else повинні передувати блоки except.
  - У блоки else можна додавати код, який потрібно запустити, якщо помилок не було.
- У прикладі наявний нескінченний цикл while.
  - Код запитує ввід даних користувачем, а потім аналізує його, використовуючи вбудовану функцію int().
  - Якщо користувач введе 0, тоді відбудеться перехід у блок except.
  - Інакше код перейде у блок else.

Оператор перехоплення винятків try-except. Застосування finally

- Якщо існує код, який потрібно запустити у всіх випадках, його пишуть у блоці finally.
  - Це найбільш поширений спосіб виконання задач з очистки.
- Нехай помилка піймана в блоці try.
  - Після переходу в блок except та його виконання будуть запущені інструкції в блоці finally.

```
try:
    x = 1 / 0
except:
    print("Error occurred")
finally:
    print("The [finally clause] is hit")
```

```
Error occurred
The [finally clause] is hit
```

- Детальніше тут.

Менеджери контексту (with statement)

- Дозволяють виділяти та вивільняти ресурси точно тоді, коли потрібно.
  - Найбільш поширений менеджер контексту – with.
  - Іноді це більш зручна конструкція, ніж try-except-finally.
  - Еквівалентні коди:

```
with open('some_file', 'w') as opened_file:
    opened_file.write('Hola!')
```

```
file = open('some_file', 'w')
try:
    file.write('Hola!')
finally:
    file.close()
```

  - Основна перевага інструкції with – забезпечення закриття файлу без уваги програміста до цього.

ДЯКУЮ ЗА УВАГУ!

Наступна тема: Структурне програмування мовою Python