Лекція 5

Оператори розгалуження й цикли в мові Python

Оператор розгалуження `if-elif-else`

Оператор розгалуження `if-elif-else` дозволяє залежно від значення логічного виразу виконати окрему ділянку програми або, навпаки, не виконати її. Оператор може використовуватись з різним набором складових в залежності від складності умови.

Найпростіший формат оператора **`if`** :

```
if <Логічний вираз>:
    4пр→<Блок, виконуваний, якщо умова дійсна>
```

Блок – це одна або більше інструкцій на мові Python, які записані підряд з однієї і тієї ж позиції.

```
if <Логічний вираз>:
4пр→<Інструкція 1>
4пр→<Інструкція 2>
4пр→<Інструкція 3>
```

Приклад 1

```
numvar = int(input("Write the number"))
firstvar, secondvar, thirdvar = 0,0,0
if numvar < 20:
    firstvar = numvar
    secondvar = numvar**2
    thirdvar = numvar**3
result = firstvar + secondvar + thirdvar
print("first_power=",numvar)
print("second_power=",secondvar)
print("third_power=",thirdvar)
print("Total=",result)
```

```
Write the number2
first_power= 2
second_power= 4
third_power= 8
Total= 14
```

Формат оператора **if-else**

```
if <Логічний вираз>:
    <Блок, виконуваний, якщо умова дійсна>
else:
    <Блок, виконуваний, якщо умова неправильна>
```

В розгорнутому вигляді цей варіант оператора **if** має вигляд:

```
if <Логічний вираз>:
    <Інструкція 1>
    <Інструкція 2>
    <Інструкція 3>
else:
    <Інструкція 4>
    <Інструкція 5>
    <Інструкція 6>
```

Приклад 2

```
numvar = int(input("Write the number: "))
if numvar < 20:
    print("floor= ",numvar // 2)
    print("remainder= ",numvar % 2)
    print("second_power= ",numvar**2)
else:
  print ("Your number is greater than 20")
```

```
Write the number: 5
floor=  2
remainder=  1
second_power=  25
```

Правила для блоків і операторів порівняння

1. Блоки усередині складової інструкції виділяються шляхом зсуву на однакову кількість пропусків (пробілів). (Зазвичай використовують чотири пробіли).

2. Ознакою кінця блоку є зсув наступної інструкції вліво щодо останньої інструкції блоку (на 4 пробіли).

3. У деяких мовах програмування логічний вираз розміщений в круглих дужках. У мові Python це робити не обов'язково, але можна, тому що будь-який вираз може бути розташований усередині круглих дужок.

Проте, круглі дужки слід використовувати тільки за необхідності розмістити умову на декількох рядках.

Для прикладу напишемо програму, яка перевіряє, чи є введене користувачем число парним.

Після перевірки виводиться відповідне повідомлення.

Приклади на правило розміщення блоків

Приклад 3.

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

Якщо блок складається з однієї інструкції, то цю інструкцію можна розмістити на одному рядку із заголовком.

Приклад 4

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

Дві інструкції в одному рядку

Якщо в одному рядку розміщено кілька інструкцій, то вони повинні відділятися одна від одної крапкою з комою.

Приклад 5

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

У цьому прикладі `end=" "` виводиться для того, щоб уникнути переходу на інший рядок

ПРИМІТКА

Не розміщуйте дві інструкції в рядку, оскільки подібна конструкція порушує стрункість коду й погіршує його супровід надалі. Завжди розміщуйте інструкцію на окремому рядку, навіть якщо блок містить тільки одну інструкцію.

Наступний код має набагато простіший і приємніший вигляд у порівнянні з попереднім:

Приклад 6.

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

Формат оператора **if…elif…else**

```
if <Логічний вираз1>:
    <Блок, виконуваний, якщо умова1 дійсна>
elif <Логічний вираз2>:
    <Блок, виконуваний, якщо умова2 дійсна>
else:
    <Блок, виконуваний, якщо всі умови неправильні>
```

Розгорнутий вигляд оператора **if-elif-else**:

```
if <Логічний вираз1>:
    <Інструкція 1>
    <Інструкція 2>
Elif <Логічний вираз2>:
    <Інструкція 3>
    <Інструкція 4>
else:
    <Інструкція 5>
    <Інструкція 6>
```

Приклад 7

```
numvar = int(input("Write the number: "))
if numvar < 20 and numvar % 2 == 0:
    print("This is an even number")
    print("floor by 3 = ",numvar // 3)
    print("second_power = ",numvar**2)
elif numvar < 20 and numvar % 2 !=0:
    print("This is an odd number")
    print("floor by 4 = ",numvar // 4)
    print("3hd power = ",numvar**3)
else:
  print ("Your number is greater than 20")
```

Загальний вигляд оператора **if…elif…else**

if <Логічний вираз>:
  $\xrightarrow{4пр}$ <Блок, виконуваний, якщо умова дійсна>
[**elif** <Логічний вираз>:
  $\xrightarrow{4пр}$ <Блок, виконуваний, якщо умова дійсна>]
[**elif** <Логічний вираз>:
  $\xrightarrow{4пр}$ <Блок, виконуваний, якщо умова дійсна>]
……
[**elif** <Логічний вираз>:
  $\xrightarrow{4пр}$ <Блок, виконуваний, якщо умова дійсна>]
[**else**:
$\xrightarrow{4пр}$ <Блок, виконуваний, якщо всі умови неправильні>

Примітка. У квадратних дужках відображені не обов'язкові складові оператора розгалуження.

Оператор `if ... else` дозволяє перевірити відразу кілька умов

Приклад 8.

```
print("""Якою операційною системою ви
користуєтесь?
1 - Windows 10
2 - Windows 8
3 - Windows 7
4 - Windows XP
5 - Інша""")
os = input("Введіть відповідне число: ")
if os == "1":
    print("Ви вибрали: Windows 10")
elif os == "2":
    print("Ви вибрали: Windows 8")
```

```
elif os == "3":
    print("Ви вибрали: Windows 7")
elif os == "4":
    print("Ви вибрали: Windows XP")
elif os == "5":
    print("Ви вибрали: інша")
elif not os:
    print("Ви не ввели число")
else:
    print("Ми не визначили вашу операційну
систему")
```

У цьому прикладі використовуються """…""" для того, щоб забезпечити багаторядковий вивід даних.

1. За допомогою інструкції **`elif`** у наведеному прикладі ми визначаємо обране значення й виводимо відповідне повідомлення.

2. Логічний вираз **`elif not os:`** не містить операторів порівняння.

- Такий запис еквівалентний наступному: `elif os == "":`
- Перевірка на рівність значенню `True` логічного виразу виконується за замовчуванням.
- Оскільки порожній рядок інтерпретується як `False`, ми інвертуємо значення, що повертається, за допомогою оператора `not`.

2. Один умовний оператор можна вкласти в інший. У цьому випадку відступ вкладеної інструкції повинен бути у два рази більший (+4 пропуски для кожної нової вкладеності)

Приклад 9.

```
print("""Якою операційною системою вы
користуєтесь?
1 - Windows 10
2 - Windows 8
3 - Windows 7
4 - Windows XP
5 - Інша""")
os = input("Введіть відповідне число: ")
if os != "":
    if os == "1":
        print("Ви вибрали: Windows 10")
    elif os == "2":
```

```
        print("Ви вибрали: Windows 8")
    elif os == "3":
        print("Ви вибрали: Windows 7")
    elif os == "4":
        print ( "Ви вибрали: Windows XP")
    elif os == "5":
        print("Ви вибрали: інша")
    else:
        print("Мы не визначили вашу операційну
систему")
else:
    print("Ви не ввели число")
```

Приклад 10.

```
print("""Якою операційною системою вы користуєтесь?
1 - Windows 10
2 - Windows 8
3 - Windows 7
4 - Windows XP
5 - Інша""")
os = input("Введіть відповідне число: ")
if not os:
    print("Ви не ввели число")
else:
    if os == "1":
        print("Ви вибрали: Windows 10")
    else:
        if os == "2":
            print("Ви вибрали: Windows 8")
        else:
            if os == "3":
                print("Ви вибрали: Windows 7")
            else:
                if os == "4":
                    print ( "Ви вибрали: Windows XP")
                else:
                    if os == "5":
                        print("Ви вибрали: інша")
```

Оператор `if ... else` має ще один формат:

```
<Змінна> = <значення> if <Умова> else
<значення>
```

Приклад 11.

```
>>> print("Yes" if 10 % 2 == 0 else "No")
Yes

>>> s = "Yes" if 10 % 2 == 0 else "No"
>>> s
'Yes'

>>> s = "Yes" if 11 % 2 == 0 else "No"
>>> s
'No'
```

Оператор циклу `for`

Припустимо, потрібно вивести всі числа від 1 до 100 по одному на рядок. Звичайним способом довелося б писати 100 рядків коду:

```
print(1)
print(2)
…
print (100)
```

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

```
for x in range(1, 101): print(x)
```

Іншими словами, цикли дозволяють виконувати ті самі інструкції багаторазово.

Формат оператора циклу `for`

Цикл **`for`** застосовується для перебору елементів послідовності й має такий формат:

`for` <Поточний елемент> **`in`** <Послідовність>:

$\xrightarrow{4пр}$<Інструкції усередині циклу>

[**`else`**:

$\xrightarrow{4пр}$<Блок, виконуваний, якщо не використовувався оператор `break`>]

Конструкції оператора `for`

```
for <Поточний елемент> in <Послідовність>:
    <Інструкції усередині циклу>
[else:<Блок без break> ]
```

- `<Послідовність>` – об'єкт, що підтримує механізм ітерації. Наприклад: рядок, список, кортеж, діапазон, словник і ін.;
- `<Поточний елемент>` – на кожній ітерації через цей параметр доступний поточний елемент послідовності або ключ словника;
- `<Інструкції усередині циклу>` – блок, який буде багаторазово виконуватися;
- якщо усередині циклу не використовувався оператор **`break`**, то після завершення виконання циклу буде виконаний блок в інструкції `else`.
- `<Блок без break>` - не є обов'язковим.

Перебір по рядку

Приклад 12. Програма перебору букв у слові

```
for s in "Я вчуся програмувати":
    print(s, end=" ")
else:
    print ( "\n Цикл виконаний")
```

```
Результат виконання:
Я  в ч у с я  п р о г р а м у в а т и
 Цикл виконаний
```

Перебір у списках і кортежах

Приклад 13. Програма перебору елементів списку

```
for x in [ "London","Paris","Washington"]:
    print(x)
```

Програма перебору елементів кортежу:

```
for y in ( "Europe", "Asia","Africa" ):
    print (y)
```

Перебір у словниках

1. Цикл `for` дозволяє також перебрати елементи словників, хоча словники й не є послідовностями.

Перший спосіб перебору використовує метод `keys()`, що повертає об'єкт `dict_keys`, який містить усі ключі словника.

Приклад 14. Використання методу `keys()`

```
>>> arr = {"x": 1, "y": 2, "z": 3}
>>> arr.keys()
dict keys (['y', 'x', 'z'] )
>>> for key in arr.keys():
        print(key, arr[key])
```

Другий спосіб. Просто вказуємо словник як параметр –

на кожній ітерації циклу буде повертатися ключ, за допомогою якого усередині циклу можна одержати значення відповідно до цього ключа.

Приклад 15.Перебір словника другим способом

```
>>> for key in arr:
        print(key, arr[key])
```

Перебір словника із сортуванням ключів

Елементи словника виводяться в довільному порядку, а не в порядку, у якім вони були зазначені при створенні об'єкта.

Щоб вивести елементи в алфавітному порядку, слід відсортувати ключі за допомогою функції `sorted ()`:

Приклад 16.

```
>>> arr = {"x":1, "y":2, "z":3}
>>> for key in sorted(arr):
    print ( key, arr [ key] )
```

Перебір елементів у складних структурах

За допомогою циклу `for` можна перебирати складні структури даних. Як приклад виведемо елементи списку кортежів.

Приклад 17.

```
>>> arr = [(1,2),(3,4)]  # Список кортежей
>>> for x, y in arr:
      print(x, y)

Результат:
1 2
3 4
```

Перебір елементів послідовності з модифікацією

Дотепер ми тільки виводили елементи послідовностей. Тепер спробуємо помножити кожний елемент списку на 2:

Приклад 18.

```
>>> s = [1,2,3,4,5,6]
>>> for i in s: print(2*i)
2
4
6
8
10
12
>>> print(s)
[1, 2, 3, 4, 5, 6]
```

Змінна i на кожній ітерації циклу містить лише копію значення поточного елемента списку.

Змінити у такий спосіб елементи списку не можна.

Перебір послідовності з використанням range()

Спосіб одержання доступу до елементів за допомогою функції **range()** шляхом генерації індексів.

Формат функція `range()`:

```
range ([<Початок>,] <Кінець> [, <Крок>])
```

1. Перший необов'язковий параметр `<Початок>`.

Задає початкове значення.

Якщо параметр не зазначений, то за замовчуванням використовується значення 0.

2. Другий обов'язковий параметр `<Кінець>` указує кінцеве значення на одиницю більше.

3. Третій необов'язковий параметр `<Крок>`

Якщо не зазначений, то використовується значення 1.

Функція повертає діапазон – особливий об'єкт, який підтримує ітераційний протокол.

Перебір зі зміною послідовності

Функція `range()` всередині циклу `for` дозволяє одержати значення поточного елемента. Змінимо послідовність шляхом множення кожного елемента списку на 2.

Приклад 19.

```
arr = [1, 2, 3]
for i in range(len(arr)):
    arr[i] *= 2
print(arr)
Результат: [2, 4, 6]
```

1.Одержуємо кількість елементів списку за допомогою функції **`len()`** і передаємо результат у функцію `range ()`.

2. Функція `range ()` повертає діапазон значень від `0` до **`len(arr)-1.`** На кожній ітерації циклу через змінну `i` доступний поточний елемент із діапазону індексів. Щоб одержати доступ до елемента списку, указуємо індекс усередині квадратних дужок. Множимо кожний елемент списку на 2, а потім виводимо результат за допомогою функції **`print().`**

Приклад використання функції `range()` .

Приклад 20. Виведемо числа від 1 до 100:

```
for i in range(1,101 ): print(i)
```

Приклад 21. Виведемо числа у зворотному порядку від 100 до 1:

```
for i in range(100, 0, -1): print(i)
```

Можна також змінювати значення не тільки на одиницю.
Приклад 22. Виведемо всі парні числа від 1 до 100:

```
for i in range(2, 101, 2): print(i)
```

Відмінність використання функції `range()` в Python 2 і Python 3

В Python 2 функція `range()` повертає список чисел.

В Python 3 функція `range()` повертає діапазон.

Щоб одержати список чисел, слід передати діапазон, що повернула функція `range()`, у функцію `list ()`.

Приклад 23.

```
>>> obj = range(len([1, 2, 3]))
>>> obj
range(0, 3)
```

```
>>> obj[0],obj[1],obj[2] # Доступ по індексу
(0, 1, 2)
>>> obj[0:2]                # Отримання зрізу
range(0, 2)

>>> i=iter(obj)
>>> next(i),next(i),next(i) # Доступ за
допомоги ітераторів
(0, 1, 2)

>>> list(obj)   # Перетворення діапазона на
список
[0, 1, 2]
>>> 1 in obj, 7 in obj   # Перевірка на
входження значення
(True, False)
```

Діапазон підтримує два корисних методи:

`index (<Значення>)` – повертає індекс елемента, що має зазначене значення. Якщо значення не входить у діапазон, виконується виключення `valueerror`.

Приклад 24.

```
>>> obj = range(1, 5)
>>> obj.index(1), obj.index(4)
(0, 3)
>>> obj.index(5)
```

….Фрагмент опущений….

```
Valueerror: 5 is not in range
```

`count (<Значення>)` – повертає кількість елементів із зазначеним значенням. Якщо елемент не входить у діапазон, повертається значення 0.

Приклад 25.

```
>>> obj = range(1, 5)
>>> obj.count(1), obj.count(10)
(1,0)
```

Функція `enumerate`

Функція має такий формат:

```
enumerate (<Об'єкт> [, start=0])
```

На кожній ітерації циклу `for` вона повертає кортеж з індексу й значення поточного елемента.

Параметр `start` може задати початкове значення індексу.

Приклад 26

Помножимо на 2 кожний елемент списку, який містить парне число.

```
arr = [1, 2, 3, 4, 5, 6]
for i, elem in enumerate(arr):
    if elem % 2 == 0:
        arr[i] *= 2
print(arr)
Результат: [1, 4, 3, 8, 5, 12]
```

Перебір послідовності за допомогою функції `enumerate()`

Функція `enumerate()` не створює список, а повертає ітератор. За допомогою функції `next ()` можна обійти всю послідовність. Коли перебір буде закінчений, виконується виключення `Stopiteration`:

Приклад 27

```
>>> arr = [1, 2]
>>> obj = enumerate(arr, start=0)
>>> next(obj)
(0, 1)
>>> next(obj)
(1, 2)
>>> next(obj)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
StopIteration
```

Оператор циклу `while`

Виконання інструкцій у циклі `while` триває доти, поки логічний вираз дійсний. Цикл `while` має наступний формат:

```
<Початкове значення>
while <Умова>:
    <Інструкції>
    <Збільшення>
[else:
    <Блок, виконуваний за умови, що не використовується оператор break>
]
```

Послідовність роботи циклу `while`:

1.Змінній-лічильнику присвоюється початкове значення.

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

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

4. Перехід до пункту 2.

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

Приклад 28

Виведемо всі числа від 1 до 100, використовуючи цикл `while`

```
i = 1                # <Початкове значення>
while i < 101:       # <Умова>
    print(i)         # <Інструкції>
    i += 1           # <Збільшення (приріст)>
```

ПРИМІТКА

Якщо `<Збільшення>` не зазначене, цикл буде нескінченним. Щоб перервати нескінченний цикл, слід натиснути комбінацію клавіш `<Ctrl>+<C>`. У результаті генерується виключення `Keyboardinterrupt`, і виконання програми зупиняється. Слід ураховувати, що перервати в такий спосіб можна тільки цикл, який виводить дані.

Приклад 29

Виведемо всі числа від 100 до 1

```
i = 100
while i:
    print(i)
    i -= 1
```

1. Тут умова не містить операторів порівняння.

2. На кожній ітерації циклу ми віднімаємо одиницю зі значення змінної-лічильника.

3. Як тільки значення буде дорівнювати 0, цикл зупиниться. Згадаємо, що число 0 у логічному контексті еквівалентно значенню `False`, а перевірка на рівність виразу значенню `True` виконується за замовчуванням.

Перебір елементів за допомогою `while`

За допомогою циклу `while` можна перебирати й елементи різних структур. Але в цьому випадку слід пам'ятати, що цикл `while` працює повільніше циклу `for`. Як приклад помножимо кожний елемент списку на 2.

Приклад 30

```
arr = [1, 2, 3]
i, count = 0, len(arr)
while i < count:
    arr[i] *= 2
    i += 1
print(arr)
```

Результат: [2, 4, 6]

Оператор `continue`

Оператор `continue` дозволяє перейти до наступної ітерації циклу до завершення виконання всіх інструкцій усередині циклу. Як приклад виведемо всі числа від 1 до 100, крім чисел від 5 до 10 включно.

Приклад 31. Застосування оператора `continue`

```
for i in range(1, 101):
    if 4 < i < 11:
        continue #Переходимо на наступну ітерацію циклу
    print(i)
```

Оператор `break`

Оператор `break` дозволяє перервати виконання циклу достроково. Для прикладу виведемо всі числа від 1 до 100 ще одним способом.

Приклад 32. Застосування оператора `break`

```
i = 1
while True:
    if i > 100: break # Перериваємо цикл
    print(i)
    i += 1
```

В умові вказано значення `True`.

У цьому випадку вираз усередині циклу має виконуватися нескінченно.

Однак використання оператора `break` перериває виконання циклу, як тільки буде надруковано 100 рядків.

ПРИМІТКА. Оператор `break` перериває виконання циклу, а не програми, тобто
далі буде виконана інструкція, що слідує відразу за циклом.

Приклад 33. Додавання невизначеної кількості чисел

```
print("Введіть слово 'stop' для одержання результату")
s = 0
while True:
    x = input("Введіть число: ")
    if x == "stop":
        break    # Вихід із циклу
    y = int(x)   # Перетворимо рядок у число
    s += y
print("Сума чисел дорівнює:", s)
```

Робота з програмою із прикладу 33.

Процес введення трьох чисел і одержання суми має такий вигляд (значення, введені користувачем, виділені напівжирним шрифтом):

```
Введіть слово 'stop' для одержання
результату

Введіть число: 10
Введіть число: 20
Введіть число: 30
Введіть число: stop
Сума чисел дорівнює: 60
```