Python: умовні оператори та цикли

Прикладна аналітика при розробці IT

Ігор Мірошниченко
КНУ імені Тараса Шевченка, ФІТ

ЗМІСТ

- Умовні оператори
- Цикли + списки
- Словники

УМОВНІ ОПЕРАТОРИ

ОПЕРАТОРИ ПОРІВНЯННЯ

- `x < y`: x строго менше y,
- `x <= y`: x менше або дорівнює y,
- `x > y`: x строго більше y,
- `x >= y`: x більше або дорівнює y,
- `x == y`: x дорівнює y,
- `x != y`: x не дорівнює y.

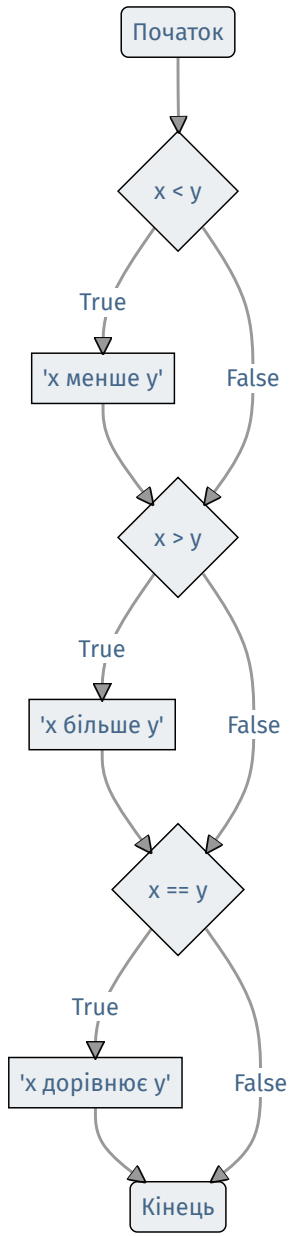
Для того, щоб ставити запитання з використанням цих символів нам знадобиться ще одне ключове слово в Python. І це ключове слово, досить просто, як і в англійській мові - `if`:

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y:
5      print('x менше y')
6  if x > y:
7      print('x більше y')
8  if x == y:
9      print('x дорівнює y')
```

```
x менше y
```

ВІЗУАЛІЗАЦІЯ КОДУ

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y:
5      print('x менше y')
6  if x > y:
7      print('x більше y')
8  if x == y:
9      print('x дорівнює y')
```



elif

Попередній код виконує свою задачу, але він не є найкращим. Якщо ви помітите, ми використовуємо три окремі запитання, щоб визначити, яке з двох чисел є більшим. Це можна зробити краще, використовуючи ключове слово `elif`.

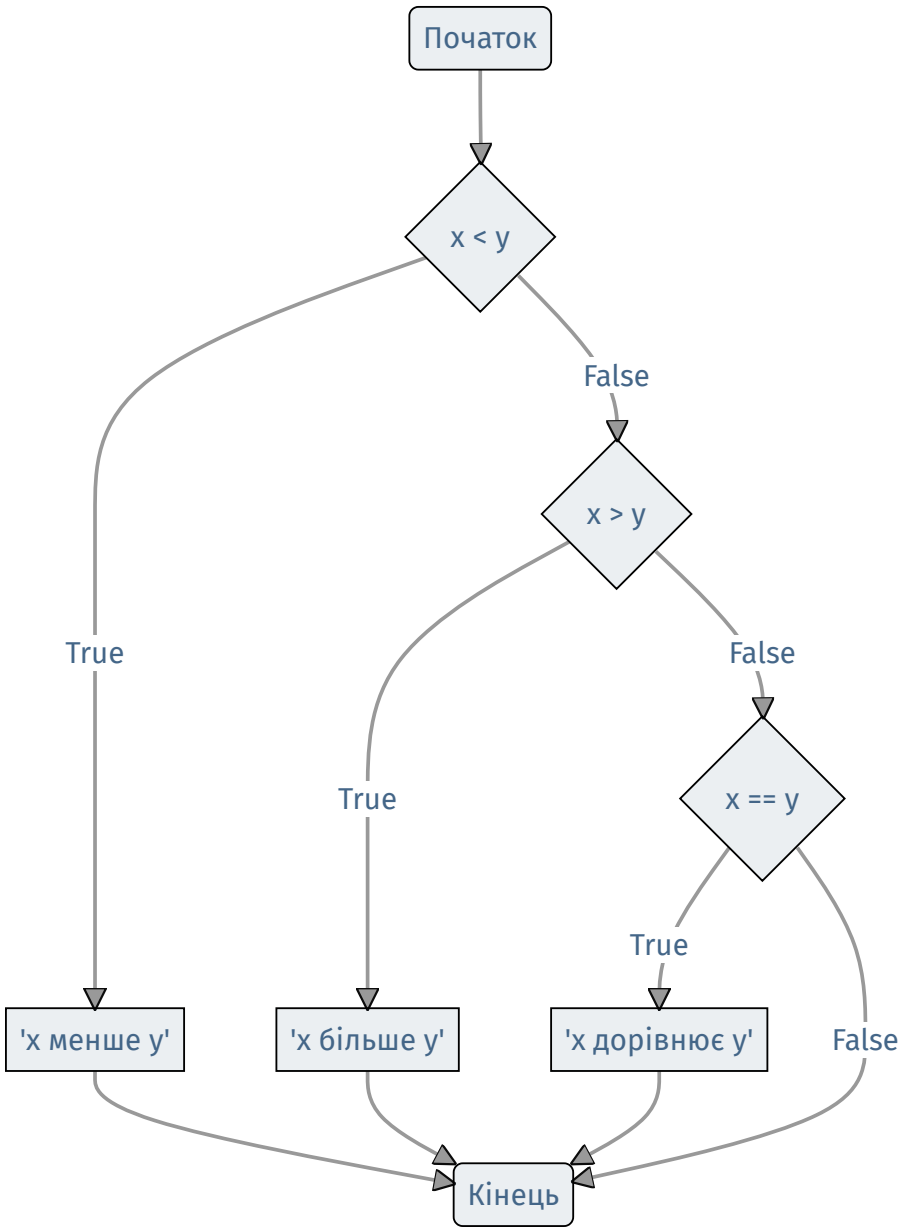
`elif` - це скорочення від `else if`.

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y:
5      print('x менше y')
6  elif x > y:
7      print('x більше y')
8  elif x == y:
9      print('x дорівнює y')
```

```
x менше y
```

ВІЗУАЛІЗАЦІЯ КОДУ

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y:
5      print('x менше y')
6  elif x > y:
7      print('x більше y')
8  elif x == y:
9      print('x дорівнює y')
```



else

Чи є сенс задавати третє, останнє питання `x == y`? Напевно ні.

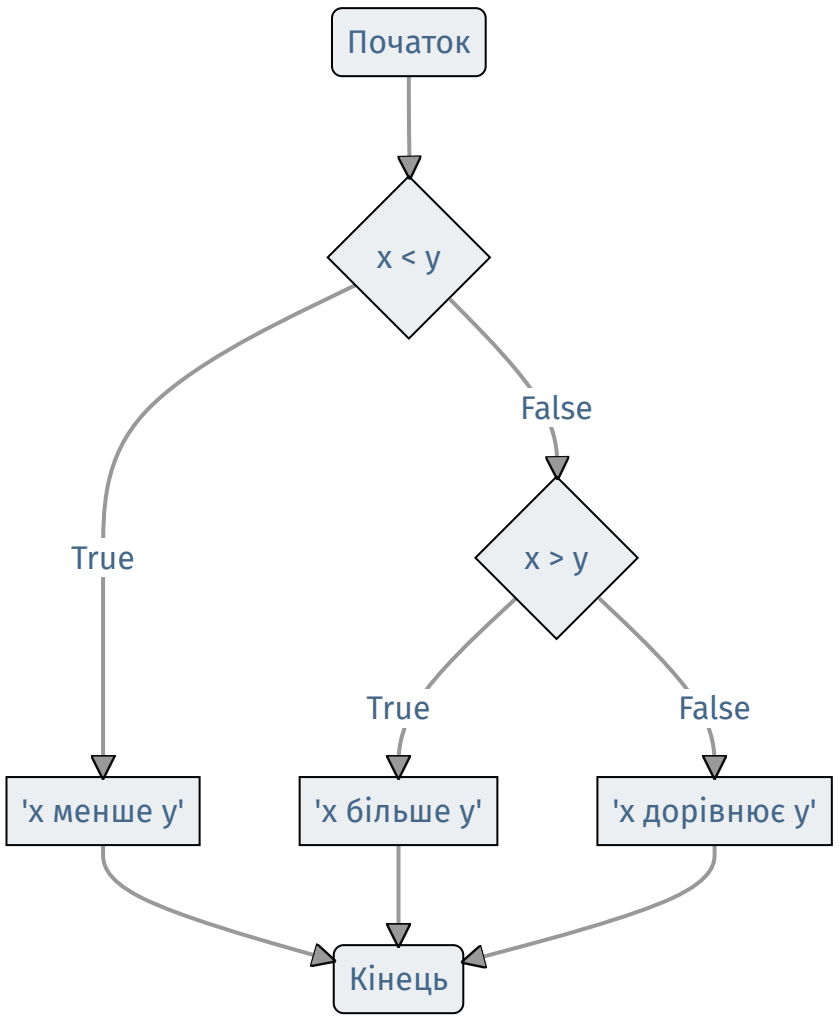
І тут нам на допомогу приходить ключове слово `else`.

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 1
3
4  if x < y:
5      print('x менше y')
6  elif x > y:
7      print('x більше y')
8  else:
9      print('x дорівнює y')
```

```
x дорівнює y
```

ВІЗУАЛІЗАЦІЯ КОДУ

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 1
3
4  if x < y:
5      print('x менше y')
6  elif x > y:
7      print('x більше y')
8  else:
9      print('x дорівнює y')
```



ЛОГІЧНИЙ ОПЕРАТОР or

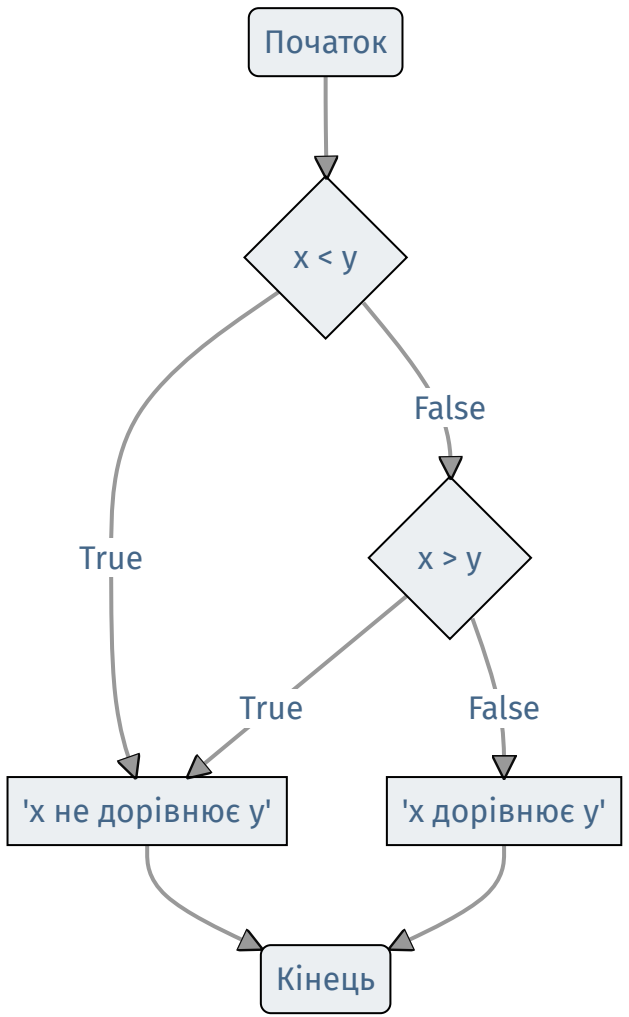
Цього разу перевіримо, чи дорівнює x числу y. Для цього використаємо оператори or:

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y or x > y:
5      print('x не дорівнює y')
6  else:
7      print('x дорівнює y')
```

```
x не дорівнює y
```

ВІЗУАЛІЗАЦІЯ КОДУ

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x < y or x > y:
5      print('x не дорівнює y')
6  else:
7      print('x дорівнює y')
```



ЛОГІЧНИЙ ОПЕРАТОР or

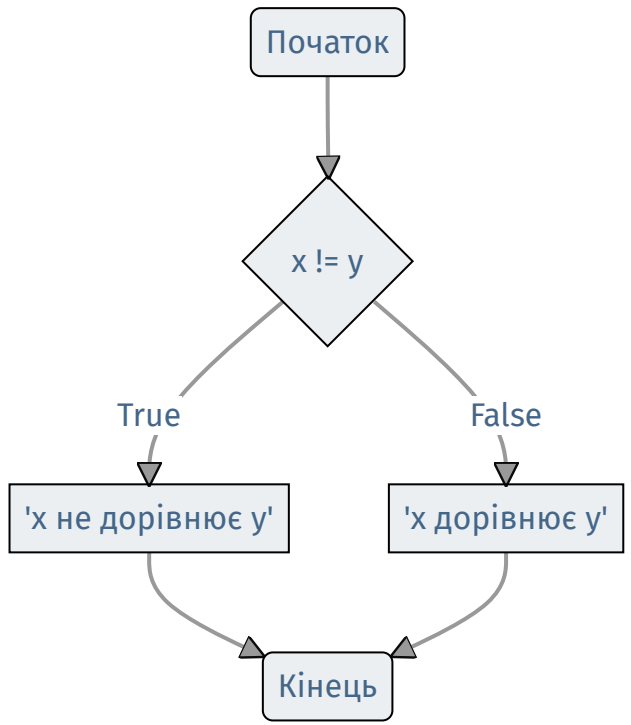
Чи можемо ми покращити цей код? Так, за допомогою оператора заперечення !=:

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x != y:
5      print('x не дорівнює y')
6  else:
7      print('x дорівнює y')
```

```
x не дорівнює y
```

ВІЗУАЛІЗАЦІЯ КОДУ

```
1  x = int(input('Введіть число x: ')) # 1
2  y = int(input('Введіть число y: ')) # 2
3
4  if x != y:
5      print('x не дорівнює y')
6  else:
7      print('x дорівнює y')
```



ЛОГІЧНИЙ ОПЕРАТОР and

Цей оператор дозволяє нам перевірити, чи виконуються обидві умови.

У змінну `score` будемо приймати значення балу. Система оцінювання в різних університетах може дещо відрізнятися, але в цілому вона виглядає так:

- `90 <= score <= 100` - A
- `80 <= score < 90` - B
- `70 <= score < 80` - C
- `66 <= score < 70` - D
- `60 <= score < 66` - E
- `21 <= score < 60` - FX
- `score < 20` - F

```
 1  score = int(input('Введіть бал: ')) # 90
 2
 3  if score >= 90 and score <= 100:
 4      print('Оцінка: A')
 5  elif score >= 80 and score < 90:
 6      print('Оцінка: B')
 7  elif score >= 70 and score < 80:
 8      print('Оцінка: C')
 9  elif score >= 66 and score < 70:
10      print('Оцінка: D')
11  elif score >= 60 and score < 66:
12      print('Оцінка: E')
13  elif score >= 21 and score < 60:
14      print('Оцінка: FX')
15  else:
16      print('Оцінка: F')
```

```
Оцінка: A
```

ЛОГІЧНИЙ ОПЕРАТОР and

Python дозволяє поміняти місцями `score` і відповідний бал при порівнянні. Такий код буде працювати так само, як і попередній:

```
 1  score = int(input('Введіть бал: ')) # 80
 2
 3  if 90 <= score and score <= 100:
 4      print('Оцінка: A')
 5  elif 80 <= score and score < 90:
 6      print('Оцінка: B')
 7  elif 70 <= score and score < 80:
 8      print('Оцінка: C')
 9  elif 66 <= score and score < 70:
10      print('Оцінка: D')
11  elif 60 <= score and score < 66:
12      print('Оцінка: E')
13  elif 21 <= score and score < 60:
14      print('Оцінка: FX')
15  else:
16      print('Оцінка: F')
```

```
Оцінка: B
```

ЛОГІЧНИЙ ОПЕРАТОР and

Або навіть об’єднувати діапазони:

```
score = int(input('Введіть бал: ')) # 70

if 90 <= score <= 100:
    print('Оцінка: A')
elif 80 <= score < 90:
    print('Оцінка: B')
elif 70 <= score < 80:
    print('Оцінка: C')
elif 66 <= score < 70:
    print('Оцінка: D')
elif 60 <= score < 66:
    print('Оцінка: E')
elif 21 <= score < 60:
    print('Оцінка: FX')
else:
    print('Оцінка: F')
```

```
Оцінка: C
```

ЛОГІЧНИЙ ОПЕРАТОР and

Якщо задуматись, то можна зробити ще краще:

```
 1  score = int(input('Введіть бал: ')) # 70
 2
 3  if score >= 90:
 4      print('Оцінка: A')
 5  elif score >= 80:
 6      print('Оцінка: B')
 7  elif score >= 70:
 8      print('Оцінка: C')
 9  elif score >= 66:
10      print('Оцінка: D')
11  elif score >= 60:
12      print('Оцінка: E')
13  elif score >= 21:
14      print('Оцінка: FX')
15  else:
16      print('Оцінка: F')
```

```
Оцінка: C
```

ПАРНІСТЬ ТА ОПЕРАТОР %

У змінну x будемо приймати значення числа. Якщо число ділиться на 2 без остачі, то воно є парним:

```
1  x = int(input('Введіть число: ')) # 2
2
3  if x % 2 == 0:
4      print('Число парне')
5  else:
6      print('Число непарне')
```

```
Число парне
```

ПАРНІСТЬ ТА ОПЕРАТОР %

Рішення через власну функцію:

```
 1  def main():
 2      x = int(input('Введіть число: ')) # 2
 3
 4      if is_even(x):
 5          print('Число парне')
 6      else:
 7          print('Число непарне')
 8
 9  def is_even(n):
10      if x % 2 == 0:
11          return True
12      else:
13          return False
14
15  main()
```

```
Число парне
```

ПАРНІСТЬ ТА ОПЕРАТОР %

А можна ще краще? Так...

```
1  def main():
2      x = int(input('Введіть число: ')) # 2
3
4      if is_even(x):
5          print('Число парне')
6      else:
7          print('Число непарне')
8
9  def is_even(n):
10     return True if n % 2 == 0 else False
11
12 main()
```

```
Число парне
```

...або ще краще:

```
1  def main():
2      x = int(input('Введіть число: ')) # 2
3
4      if is_even(x):
5          print('Число парне')
6      else:
7          print('Число непарне')
8
9  def is_even(n):
10     return n % 2 == 0
11
12 main()
```

```
Число парне
```

ОПЕРАТОР match

match - це механізм зіставлення з шаблонами, дозволяє вам виконувати дії в залежності від значення змінної.

Напишемо код, який запитує у користувача його ім’я, а потім просто виводить гуртожиток, на якому він навчається у книгах про Гаррі Поттера:

```
 1  name = input('Введіть ім\'я: ') # Драко
 2
 3  if name == 'Гаррі':
 4      print('Ґрифіндор')
 5  elif name == 'Герміона':
 6      print('Ґрифіндор')
 7  elif name == 'Рон':
 8      print('Ґрифіндор')
 9  elif name == 'Драко':
10      print('Слизерин')
11  else:
12      print('Хто?')
```

```
Слизерин
```

ОПЕРАТОР match

У попередньому коді є певна повторюваність: Гаррі, Герміона та Рон належать до Гріфіндору. Давайте об’єднаємо ці умови в одну:

```
1  name = input('Введіть ім\'я: ') # Герміона
2
3  if name == 'Гаррі' or name == 'Герміона' or name == 'Рон':
4      print('Ґрифіндор')
5  elif name == 'Драко':
6      print('Слизерин')
7  else:
8      print('Хто?')
```

```
Ґрифіндор
```

ОПЕРАТОР match

А тепер, нарешті, через match:

```
 1  name = input('Введіть ім\'я: ') # Рон
 2
 3  match name:
 4      case 'Гаррі':
 5          print('Ґрифіндор')
 6      case 'Герміона':
 7          print('Ґрифіндор')
 8      case 'Рон':
 9          print('Ґрифіндор')
10      case 'Драко':
11          print('Слизерин')
12      case _:
13          print('Хто?')
```

```
Ґрифіндор
```

Зверніть увагу на те, що я використовую _ для визначення варіанту за замовчуванням. Це означає, що якщо жоден з варіантів не відповідає значенню name, то виконається варіант за замовчуванням. Це дозволяє нам уникнути використання else в кінці.

ОПЕРАТОР match

Можна краще? Так - об’єднання трьох операторів case ми можемо використовувати оператор |:

```
1  name = input('Введіть ім\'я: ') # Гаррі
2
3  match name:
4      case 'Гаррі' | 'Герміона' | 'Рон':
5          print('Ґрифіндор')
6      case 'Драко':
7          print('Слизерин')
8      case _:
9          print('Хто?')
```

```
Ґрифіндор
```

Варто зазначити, що Вам не потрібен оператор break для виходу з case, Ви просто використовуєте _ як паличку-виручалочку в кінці коду.

ПОРІВНЯННЯ РЯДКІВ

Порівняння рядків влаштовано трохи хитріше. Почнемо з простого та розглянемо порівняння двох однакових рядків:

```
1  print('abc' == 'abc')
```

```
True
```

А якщо записати так:

```
1  print('abc' > 'abc')
```

```
False
```

Порівняння рядків у Python відбувається лексикографічно, тобто посимвольно зліва направо. Якщо символи збігаються, Python переходить до наступного символу в рядку і продовжує порівняти доти, доки не знайде різниця між символами в рядку.

```
1  str1 = 'hello'
2  str2 = 'world'
3  print(str1 < str2 )
```

```
True
```

```
1  str1 = 'apple'
2  str2 = 'banana'
3  print(str1 > str2)
```

```
False
```

```
1  str1 = 'abc'
2  str2 = 'aba'
3  print(str1 > str2 )
```

```
True
```

ЦИКЛИ + СПИСКИ

ЦИКЛ `while`

Припустимо нам потрібно створити програму кота, який буде “нявкти” три рази:

```
1  print("Няв!")
2  print("Няв!")
3  print("Няв!")
```

```
Няв!
Няв!
Няв!
```

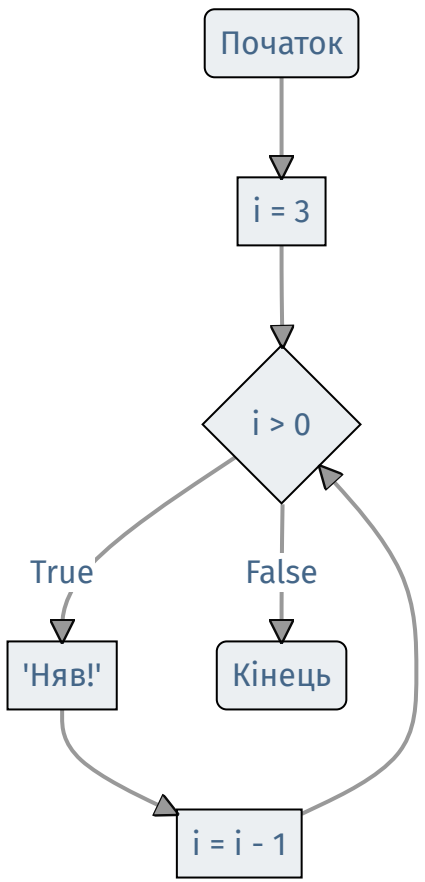
Якщо масштабувати задачу, то вочевидь це не найкращій варіант. Для цього існують цикли.

ЦИКЛ `while`

Ключове слово `while` - це конструкція, яка дозволяє ставити питання знову і знову.

```
1  i = 3
2  while i > 0:
3      print("Няв!")
4      i = i - 1
```

```
Няв!
Няв!
Няв!
```

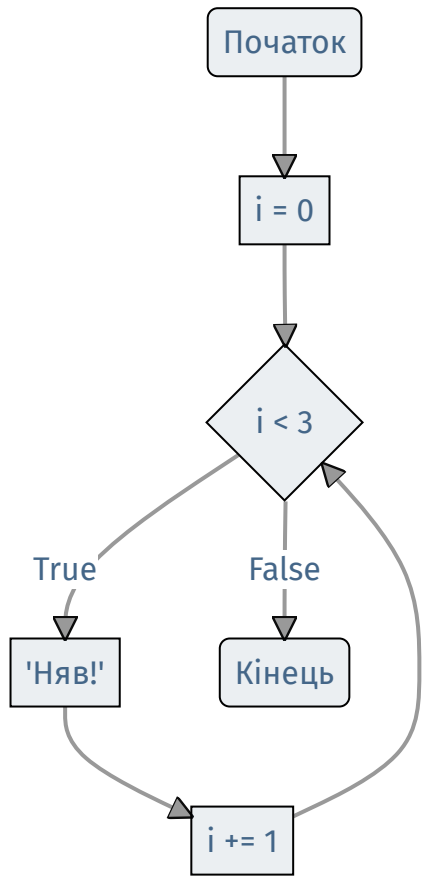


ЦИКЛ `while`

Або можна піти від зворотного і використати “синтаксичний цукор”:

```
1  i = 0
2  while i < 3:
3      print("Няв!")
4      i += 1
```

```
Няв!
Няв!
Няв!
```



ЦИКЛ `for` ТА СПИСКИ

`list` - це структура даних, яка дозволяє зберігати декілька значень у одній змінній.

Списки в Python створюються за допомогою квадратних дужок `[]` і можуть містити будь-які значення, в тому числі інші списки.

Принцип роботи циклу `for` полягає в тому, що він дозволяє ітераційно перебирати список елементів, наприклад так:

```
1  for i in [0, 1, 2]:
2      print("Няв!")
```

```
Няв!
Няв!
Няв!
```

ЦИКЛ for ТА СПИСКИ

Що в попередньому варіанті коду не так? Якщо ми захочемо змінити кількість нявкань, нам доведеться змінювати список. Це не дуже зручно. Щоб цього уникнути, ми можемо скористатися функцією `range()`, яка дозволяє створити список послідовності чисел:

```
1  for i in range(3):
2      print("Няв!")
```

```
Няв!
Няв!
Няв!
```

Pythonic варіант:

```
1  for _ in range(3):
2      print("Няв!")
```

```
Няв!
Няв!
Няв!
```

А можна ще?

```
1  print("Няв!\n" * 3, end="")
```

```
Няв!
Няв!
Няв!
```

while + for

Давайте запитаємо користувача, скільки разів цей кіт має нявкнути:

```
1  while True:
2      n = int(input("Скільки разів кіт має нявкнути? ")) # 3
3          if n > 0:
4              break
5
6  for _ in range(n):
7      print("Няв!")
```

```
Няв!
Няв!
Няв!
```

while + for + ВЛАСНА ФУНКЦІЯ

Давайте ще трошки попрактикуємо писати власні функції:

```
 1  def main():
 2      number = get_number()
 3      cat(number)
 4
 5  def get_number():
 6      while True:
 7          n = int(input("Скільки разів кіт має нявкнути? "))
 8          if n > 0:
 9              break
10      return n
11
12  def cat(n):
13      for _ in range(n):
14          print("Няв!")
15
16  main()
```

ІНДЕКСАЦІЯ СПИСКІВ

Потренуємося працювати з індексами списків.

До будь-якого елемента списку можна звернутися за його індексом. Запам'ятайте, що індексація починається з нуля:

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
```

Як роздрукувати список студентів? Для цього я можу скористатися індексом, який вказує на позицію елемента у списку за допомогою квадратних [] дужок після назви змінної:

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  print(students[0])
4  print(students[1])
5  print(students[2])
```

```
Гаррі Поттер
Герміона Грейнджер
Рон Уізлі
```

ІНДЕКСАЦІЯ СПИСКІВ

Але ж має бути кращий спосіб, чи не так?

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  for student in students:
4      print(student)
```

```
Гаррі Поттер
Герміона Грейнджер
Рон Уізлі
```

А що, якщо я хочу крім імені студента вивести його номер у списку? Для цього я можу скористатися функцією `enumerate()`, яка дозволяє мені отримати індекс елемента у списку:

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  for i, student in enumerate(students):
4      print(f"{i + 1}: {student}")
```

```
1: Гаррі Поттер
2: Герміона Грейнджер
3: Рон Уізлі
```

ІНДЕКСАЦІЯ СПИСКІВ

Можна робити зрізи списків:

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  print(students[0:2])
```

```
['Гаррі Поттер', 'Герміона Грейнджер']
```

МЕТОДИ СПИСКІВ

Методи - це функції, які можна викликати для списків. Методи списків дозволяють виконувати різні дії зі списками.

- `append()` - додає елемент у кінець списку

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  students.append("Драко Малфой")
4  print(students)
```

```
['Гаррі Поттер', 'Герміона Грейнджер', 'Рон Уізлі', 'Драко Малфой']
```

- `clear()` - видаляє всі елементи списку

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  students.clear()
4  print(students)
```

```
[]
```

- `copy()` - повертає копію списку

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  students_copy = students.copy()
4  print(students_copy)
```

```
['Гаррі Поттер', 'Герміона Грейнджер', 'Рон Уізлі']
```

МЕТОДИ СПИСКІВ

- `count()` - повертає кількість елементів списку

```
1  house = ["Ґрифіндор", "Ґрифіндор", "Ґрифіндор", "Слизерин"]
2
3  print(house.count("Слизерин"))
```

```
1
```

- `extend()` - додає елементи іншого списку до поточного

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  new_students = ["Драко Малфой", "Невілл Лонгботтом"]
4  print(students.extend(new_students))
```

```
None
```

- `index()` - повертає індекс першого елемента зі списку

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  print(students.index("Герміона Грейнджер"))
```

```
1
```

МЕТОДИ СПИСКІВ

- `insert()` - додає елемент у список за індексом

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі"]
2
3  students.insert(1, "Драко Малфой")
4  print(students)
```

```
['Гаррі Поттер', 'Драко Малфой', 'Герміона Грейнджер', 'Рон Уізлі']
```

- `pop()` - видаляє та повертає елемент із заданим індексом

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі", "Драко Малфой"]
2
3  removed_student = students.pop(1)
4  print(removed_student)
5  print(students)
```

```
Герміона Грейнджер
['Гаррі Поттер', 'Рон Уізлі', 'Драко Малфой']
```

- `remove()` - видаляє елемент із списку за значенням

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі", "Драко Малфой"]
2
3  students.remove("Драко Малфой")
4  print(students)
```

```
['Гаррі Поттер', 'Герміона Грейнджер', 'Рон Уізлі']
```

МЕТОДИ СПИСКІВ

- `reverse()` - змінює порядок елементів у списку на зворотний

```
1  students = ["Гаррі Поттер", "Герміона Грейнджер", "Рон Уізлі", "Драко Малфой"]
2
3  students.reverse()
4  print(students)
```

```
['Драко Малфой', 'Рон Уізлі', 'Герміона Грейнджер', 'Гаррі Поттер']
```

- `sort()` - сортує елементи списку

```
1  students = ["Герміона Грейнджер", "Гаррі Поттер", "Драко Малфой", "Рон Уізлі"]
2
3  students.sort()
4  print(students)
```

```
['Гаррі Поттер', 'Герміона Грейнджер', 'Драко Малфой', 'Рон Уізлі']
```

LIST COMPREHENSION

List comprehension в Python - це компактний спосіб створення нового списку на основі наявного списку або іншої ітерованої послідовності, наприклад, рядка.

Загальний синтаксис для створення list comprehension виглядає наступним чином:

```
1  new_list = [expression for item in iterable if condition]
```

де:

- `expression` - вираз, який застосовуватиметься до кожного елемента списку або послідовності
- `item` - змінна, що представляє поточний елемент послідовності, що ітерується;
- `iterable` - список або інша послідовність, що ітерується;
- `condition` (необов'язково) - умова, за якою буде фільтруватися вихідна послідовність.

LIST COMPREHENSION

1. Припустимо, у нас є список із числами, ми хочемо створити новий список із квадратами цих чисел. Варіант рішення:

```
1  numbers = [1, 2, 3, 4, 5]
2  squares = []
3  for i in numbers:
4      squares.append(i**2)
5
6  print(squares)
```

```
[1, 4, 9, 16, 25]
```

А ось так виглядало б рішення за допомогою list comprehension:

```
1  numbers = [1, 2, 3, 4, 5]
2  squares = [x ** 2 for x in numbers]
3
4  print(squares)
```

```
[1, 4, 9, 16, 25]
```

Інакше кажучи, результат роботи нашого циклу відразу поміщається в список.

LIST COMPREHENSION

2. Створення нового списку, що містить тільки парні числа з вихідного списку:

Стандартний розв'язок:

```
1  numbers = [1, 2, 3, 4, 5]
2  even_numbers = []
3  for i in numbers:
4      if i % 2 == 0:
5          even_numbers.append(i)
6
7  print(even_numbers)
```

```
[2, 4]
```

List comprehension:

```
1  numbers = [1, 2, 3, 4, 5]
2  even_numbers = [x for x in numbers if x % 2 == 0]
3
4  print(even_numbers)
```

```
[2, 4]
```

LIST COMPREHENSION

3. Створення нового списку, що містить довжини слів із вихідного списку:

```
1  words = ["hello", "world", "python", "list"]
2  word_lengths = [len(word) for word in words]
3
4  print(word_lengths)
```

```
[5, 5, 6, 4]
```

СЛОВНИКИ

СЛОВНИКИ

Словники - це структура даних, яка дозволяє зберігати пари ключ-значення у одній змінній. Словники в Python створюються за допомогою фігурних дужок {} і можуть містити будь-які значення, в тому числі інші словники.

Припустимо, що ми хочемо відстежувати, хто до якого гуртожитку відноситься в Гоґвортсі. Для прикладу візьмемо чотирьох учнів:

Герміона Грейнджер	Гаррі Поттер	Рон Уізлі	Драко Малфой
Ґрифіндор	Ґрифіндор	Ґрифіндор	Слизерин

```
1  students = {
2      "Герміона Грейнджер": "Ґрифіндор",
3      "Гаррі Поттер": "Ґрифіндор",
4      "Рон Уізлі": "Ґрифіндор",
5      "Драко Малфой": "Слизерин"
6  }
7
8  print(students)
```

```
{'Герміона Грейнджер': 'Ґрифіндор', 'Гаррі Поттер': 'Ґрифіндор', 'Рон Уізлі': 'Ґрифіндор', 'Драко Малфой': 'Слизерин'}
```

СЛОВНИКИ

Тепер давайте виведемо на екран ім'я та гуртожиток кожного учня. Для цього ми можемо скористатися циклом `for`:

```
1  students = {
2      "Герміона Грейнджер": "Ґрифіндор",
3      "Гаррі Поттер": "Ґрифіндор",
4      "Рон Уізлі": "Ґрифіндор",
5      "Драко Малфой": "Слизерин"
6  }
7
8  for student in students:
9      print(student)
```

```
Герміона Грейнджер
Гаррі Поттер
Рон Уізлі
Драко Малфой
```

Щоб вивести ім'я та гуртожиток кожного учня, ми можемо скористатися індексацією словника за ключем:

```
1  students = {
2      "Герміона Грейнджер": "Ґрифіндор",
3      "Гаррі Поттер": "Ґрифіндор",
4      "Рон Уізлі": "Ґрифіндор",
5      "Драко Малфой": "Слизерин"
6  }
7
8  for student in students:
9      print(f"{student}: {students[student]}")
```

```
Герміона Грейнджер: Ґрифіндор
Гаррі Поттер: Ґрифіндор
Рон Уізлі: Ґрифіндор
Драко Малфой: Слизерин
```

СЛОВНИКИ

Давайте ускладнимо задачу і додамо ще одину характеристику до кожного учня - його Патронуса:

Ім’я	Гуртожиток	Патронус
Герміона Грейнджер	Ґрифіндор	Видра
Гаррі Поттер	Ґрифіндор	Олень
Рон Уізлі	Ґрифіндор	Тер’єр
Драко Малфой	Слизерин	

```
students = [
    {
        "ім'я": "Герміона Грейнджер",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Видра"
    },
    {
        "ім'я": "Гаррі Поттер",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Олень"
    },
    {
        "ім'я": "Рон Уізлі",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Тер'єр"
    },
    {
        "ім'я": "Драко Малфой",
        "гуртожиток": "Слизерин",
        "патронус": None
    }
]
```

Зверніть увагу, що згідно канонічного варіанту Драко Малфой не має Патронуса. Тому ми вказали його значення як None - це спеціальне значення, яке означає “нічого”.

СЛОВНИКИ

Для виводу інформації про учнів ми можемо скористатися циклом `for`:

```
students = [
    {
        "ім'я": "Герміона Грейнджер",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Видра"
    },
    {
        "ім'я": "Гаррі Поттер",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Олень"
    },
    {
        "ім'я": "Рон Уізлі",
        "гуртожиток": "Ґрифіндор",
        "патронус": "Тер'єр"
    },
    {
        "ім'я": "Драко Малфой",
        "гуртожиток": "Слизерин",
        "патронус": None
    }
]

for student in students:
    print(student["ім'я"], student["гуртожиток"], student["патронус"], sep=", ")
```

```
Герміона Грейнджер, Ґрифіндор, Видра
Гаррі Поттер, Ґрифіндор, Олень
Рон Уізлі, Ґрифіндор, Тер'єр
Драко Малфой, Слизерин, None
```

СЛОВНИКИ

Альтернативний варіант створення словника `students` - це використання вкладених словників:

```
students = {
    "Герміона Грейнджер": {
        "гуртожиток": "Ґрифіндор",
        "патронус": "Видра"
    },
    "Гаррі Поттер": {
        "гуртожиток": "Ґрифіндор",
        "патронус": "Олень"
    },
    "Рон Уізлі": {
        "гуртожиток": "Ґрифіндор",
        "патронус": "Тер'єр"
    },
    "Драко Малфой": {
        "гуртожиток": "Слизерин",
        "патронус": None
    }
}

for student in students:
    print(student, students[student]["гуртожиток"], students[student]["патронус"], sep=", ")
```

```
Герміона Грейнджер, Ґрифіндор, Видра
Гаррі Поттер, Ґрифіндор, Олень
Рон Уізлі, Ґрифіндор, Тер'єр
Драко Малфой, Слизерин, None
```

СТВОРЕННЯ СЛОВНИКІВ

У Python є кілька способів створення словників:

1. Літерали словника

```
1  my_dict = {'ім\'я': 'Аліса', 'вік': 25, 'місто': 'Київ'}
2  print(my_dict)
```

```
{"ім'я": 'Аліса', 'вік': 25, 'місто': 'Київ'}
```

2. Вбудована функція `dict()`

```
1  empty_dict = dict()
2  print(empty_dict)
```

```
{}
```

```
1  fruit_dict = dict([('яблуко', 5), ('банан', 3), ('апельсин', 2)])
2  print(fruit_dict)
```

```
{'яблуко': 5, 'банан': 3, 'апельсин': 2}
```

СТВОРЕННЯ СЛОВНИКІВ

3. **Генератор словників**

Генератори словників дозволяють створювати словники на основі інших послідовностей. Наприклад:

```
1  fruit_list = ['яблуко', 'банан', 'апельсин']
2  fruit_dict = {fruit: len(fruit) for fruit in fruit_list}
3  print(fruit_dict)
```

```
{'яблуко': 6, 'банан': 5, 'апельсин': 8}
```

4. **Метод `fromkeys()`**

Метод `fromkeys()` дає змогу створити словник із зазначеними ключами й одним і тим самим значенням для всіх ключів. Наприклад:

```
1  fruit_list = ['яблуко', 'банан', 'апельсин']
2  fruit_dict = {}.fromkeys(fruit_list, 0)
3  print(fruit_dict)
```

```
{'яблуко': 0, 'банан': 0, 'апельсин': 0}
```

МЕТОДИ СЛОВНИКІВ

1. **`get()`** - дає змогу отримати значення за ключем, але на відміну від звернення через `[]`, не викликає виняток `KeyError`, якщо ключ відсутній у словнику:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  print(my_dict.get("two"))
3  print(my_dict.get("four", "Key not found"))
```

```
2
Key not found
```

2. **`keys()`** - дає змогу отримати список ключів словника:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  print(my_dict.keys())
```

```
dict_keys(['one', 'two', 'three'])
```

3. **`values()`** - дає змогу отримати список значень словника:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  print(my_dict.values())
```

```
dict_values([1, 2, 3])
```

МЕТОДИ СЛОВНИКІВ

4. `items()` - дає змогу отримати список пар ключ-значення словника:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  print(my_dict.items())
```

```
dict_items([('one', 1), ('two', 2), ('three', 3)])
```

5. `pop()` - дає змогу видалити пару ключ-значення за ключем. Якщо ключ відсутній, викликає виняток `KeyError` або повертає другий аргумент, якщо він переданий:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  print(my_dict.pop("two"))
3  print(my_dict.pop("four", "Key not found"))
```

```
2
Key not found
```

6. `clear()` - дає змогу видалити всі пари ключ-значення зі словника:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  my_dict.clear()
3  print(my_dict)
```

```
{}
```

МЕТОДИ СЛОВНИКІВ

7. **update()** - дає змогу оновити словник із іншого словника або із послідовності пар ключ-значення:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  new_dict = {"two": 22, "four": 4}
3  my_dict.update(new_dict)
4  print(my_dict)
```

```
{'one': 1, 'two': 22, 'three': 3, 'four': 4}
```

8. **copy()** - дає змогу створити копію словника:

```
1  my_dict = {"one": 1, "two": 2, "three": 3}
2  new_dict = my_dict.copy()
3  print(new_dict)
```

```
{'one': 1, 'two': 2, 'three': 3}
```

ОПЕРАТОРИ `continue` ТА `break`

Оператор `continue` і оператор `break` є керуючими операторами в циклах, їх використовують для зміни поведінки виконання циклу.

Оператор `continue` використовується для пропуску частини тіла циклу, що залишилася, і переходу до наступної ітерації циклу. Якщо зустрінеться оператор `continue`, то код нижче за нього в поточній ітерації циклу не буде виконано, а виконання циклу продовжиться відразу з наступної ітерації:

```
1  for i in range(1, 6):
2      if i == 3:
3          continue
4      print(i)
```

```
1
2
4
5
```

ОПЕРАТОРИ `continue` ТА `break`

Оператор `break` використовується для переривання виконання циклу. Якщо зустрінеться оператор `break`, виконання циклу буде припинено і відбувається вихід із циклу:

```
1  for i in range(1, 6):
2      if i == 3:
3          break
4      print(i)
```

```
1
2
```

ОПЕРАТОРИ continue ТА break

Приклад спільного використання:

```python
employees = [
    {'name': 'John', 'qualification': 'low'},
    {'name': 'Alice', 'qualification': 'high'},
    {'name': 'Bob', 'qualification': 'medium'},
    {'name': 'Eva', 'qualification': 'low'},
    {'name': 'Mike', 'qualification': 'high'},
    {'name': 'Lisa', 'qualification': 'medium'}
]

found_high_qualification_employee = False

for employee in employees:
    qualification = employee['qualification']
    if qualification == 'low':
        continue # Пропускаємо працівника з низькою кваліфікацією
    elif qualification == 'high':
        found_high_qualification_employee = True
        high_qualification_employee = employee
        break # Знайдено працівника з високою кваліфікацією, припиняємо пошук
if found_high_qualification_employee:
    print("Працівника з високою кваліфікацією знайдено!")
    print(high_qualification_employee)
else:
    print("Працівника з високою кваліфікацією не знайдено.")
```

```
Працівника з високою кваліфікацією знайдено!
{'name': 'Alice', 'qualification': 'high'}
```

ДЯКУЮ ЗА УВАГУ!

Матеріали курсу

ihor.miroshnychenko@kneu.ua

Data Mirosh

@ihormiroshnychenko

@aranaur

aranaur.rbind.io