**Модуль 2: Керуючі конструкції. Винятки**

name = input("What is your name? ")  
print(f"Hello {name}")  
За замовчуванням у Python інструкції виконуються одна за одною зверху вниз. У прикладі дві інструкції — спочатку виконується name = input("What is your name? "), потім print(f"Hello {name}").

Послідовність виконання виразів в програмі називається «потік виконання» (Flow of execution).

У Python існує три способи управління потоком виконання:

* умовне виконання — виконання блоку інструкцій тільки при настанні деякої умови;
* цикли — повторення виконання блоку інструкцій, доки виконується деяка умова;
* винятки — виконання блоку інструкцій у разі помилки.

**Умовне виконання**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%83%D0%BC%D0%BE%D0%B2%D0%BD%D0%B5-%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D0%B0%D0%BD%D0%BD%D1%8F)

age = input("How old are you? ")  
  
if int(age) >= 18:  
 print("You are adult already.")  
else:  
 print("You are infant yet.")

У Python реалізований оператор контролю виконання (умовний оператор) if ... elif ... else.

Оператор контролю виконання дозволяє виконувати блоки інструкцій не завжди, а тільки тоді, коли буде виконана умова.

Синтаксис умовного оператора:

* починається з ключового слова if, за яким йде умова;
* після умови ставиться двокрапка і з нового рядка з відступом йде блок інструкцій, які будуть виконані, якщо умова виконується;
* після блоку if може бути нуль або більше блоків elif, інтерпретатор послідовно перевірятиме усі умови elif зверху вниз, доки не знайде той, який виконується;
* потім може бути один блок else, який виконується, якщо всі попередні умови не виконуються.

a = input('Введіть число')  
a = int(a)  
if a > 0:  
 print('Число додатне')  
elif a < 0:  
 print("Число від'ємне")  
else:  
 print('Це число - нуль')

Під час виконання умовного оператора інтерпретатор Python перевіряє умови зверху вниз, доки не знайде те, яке виконується, потім виконає вираз для цієї умови та вийде з перевірки умов.

a = input('Введіть число')  
a = int(a)  
if a > 0:  
 print('Число додатне')  
elif a == 1:  
 print('Число дорівнює 1')  
else:  
 print("a <= 0")

В такому випадку код для умови a == 1 ніколи не виконається.

**Умови у Python**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%83%D0%BC%D0%BE%D0%B2%D0%B8-%D1%83-python)

Умовний оператор if ... elif ... else у Python у якості умов може приймати змінні типу bool або будь-який вираз, який він виконає і результат перетворить в bool.

**Логічні вирази**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D1%96-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7%D0%B8)

Коли в якості умови в умовний оператор ми передаємо вираз, то вираз виконається, а результат його виконання буде перетворений в тип bool.

Для зручності у Python є механізм неявного приведення будь-якого типу до типу bool. Правила приведення до bool — інтуїтивні:

* число 0 приводиться до False (ціле, дробове або комплексне);

money = 0  
if money:  
 print(f"You have {money} on your bank account")  
else:  
 print("You have no money and no debts")

* None приводиться до False;

result = None  
if result:  
 print(result)  
else:  
 print("Result is None, do something")

* порожній контейнер (порожній рядок тощо) приводиться до False

user\_name = input("Enter your name: ")  
  
if user\_name:  
 print(f"Hello {user\_name}")  
else:  
 print("Hi Anonym!")

* все інше приводиться до True

Правила приведення до bool дозволяють писати умовні вирази у Python практично літературною англійською. В будь-якому разі, такий код стає дуже зрозумілим.

**Булева алгебра**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%B1%D1%83%D0%BB%D0%B5%D0%B2%D0%B0-%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0)

Що робити, якщо у нас складна умова, яка поєднує в собі декілька вкладених умов? Наприклад, щоб користувач міг орендувати автомобіль, потрібно, щоб у користувача обов'язково було вказане ім'я, користувач був старший 18 і були водійські права.

name = "Taras"  
age = 22  
has\_driver\_licence = True  
  
if name and age >= 18 and has\_driver\_licence:  
 print(f"User {name} can rent a car")

Для побудови логічних умов з декількох, використовується булева алгебра.

**Булева алгебра — це розділ математичної логіки, в якому вивчаються логічні операції над висловлюваннями.**

У програмуванні застосовують бінарну логіку, можливі значення для бінарної логіки можуть бути True та False.

Булева алгебра будується на трьох основних операціях: "І", "АБО", "НЕ". Є ще допоміжні, але давайте поки що розглянемо основні.

У Python оператори булевої алгебри — це оператори not, and, or.

**and (І) вираз виконується, якщо обидві умови виконуються**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#and-%D1%96-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7-%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D1%83%D1%94%D1%82%D1%8C%D1%81%D1%8F-%D1%8F%D0%BA%D1%89%D0%BE-%D0%BE%D0%B1%D0%B8%D0%B4%D0%B2%D1%96-%D1%83%D0%BC%D0%BE%D0%B2%D0%B8-%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D1%83%D1%8E%D1%82%D1%8)

| **A** | **B** | **A and B** |
| --- | --- | --- |
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |

a = True and False *# False*

**or (АБО) вираз виконується, якщо хоча б одна з умов виконується**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#or-%D0%B0%D0%B1%D0%BE-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7-%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D1%83%D1%94%D1%82%D1%8C%D1%81%D1%8F-%D1%8F%D0%BA%D1%89%D0%BE-%D1%85%D0%BE%D1%87%D0%B0-%D0%B1-%D0%BE%D0%B4%D0%BD%D0%B0-%D0%B7-%D1%83%D0%BC%D0%BE%D0%B2-%D0%B2%D0%B8%D0%BA%D0)

| **A** | **B** | **A or B** |
| --- | --- | --- |
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |

a = True or False *# True*

**not (НІ) заперечення, вираз виконується, якщо операнд — неправда**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#not-%D0%BD%D1%96-%D0%B7%D0%B0%D0%BF%D0%B5%D1%80%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7-%D0%B2%D0%B8%D0%BA%D0%BE%D0%BD%D1%83%D1%94%D1%82%D1%8C%D1%81%D1%8F-%D1%8F%D0%BA%D1%89%D0%BE-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D0%BD%D0%B4--%D0%)

| **A** | **not A** |
| --- | --- |
| True | False |
| False | True |

a = not 2 < 0 *# True*

**Тернарні операції**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%80%D0%BD%D1%96-%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97)

Тернарні оператори — це ті самі умовні вирази, але в скороченій формі. Ці оператори повертають щось, залежно від того, чи є умова істиною або брехнею.

is\_nice = True  
state = "nice" if is\_nice else "not nice"

У цьому прикладі у state буде рядок 'nice'.

Такий підхід дозволяє швидко перевірити умову, а не писати декілька рядків оператора if ... else ....

У Python також існує коротший варіант тернарного оператора.

some\_data = None  
msg = some\_data or "Не було повернено даних"

У цьому прикладі msg містить рядок 'Не було повернено даних', це зручно, коли потрібно швидко перевірити значення та показати повідомлення, якщо значення None.

**Зверніть увагу**, що для скороченої форми використовується саме оператор or (**АБО**).

**Блоки інструкцій**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%B1%D0%BB%D0%BE%D0%BA%D0%B8-%D1%96%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D1%96%D0%B9)

x = int(input("X: "))  
y = int(input("Y: "))  
  
if x == 0:  
 print("X can`t be equal to zero")  
 x = int(input("X: "))  
  
result = y / x

У Python особливий синтаксис стосовно виділення блоків інструкцій. Щоб інтерпретатор сприйняв набір інструкцій як окремий блок, достатньо виділити всі інструкції цього блоку однаковою кількістю відступів зліва. У Python рекомендується для виділення одного рівня вкладеності для блоку інструкцій використовувати **4 пробіли**.

Ви можете використати символи табуляції для виділення блоку інструкцій, це не помилка, але такий спосіб не рекомендується.

Синтаксичною **помилкою** буде змішати в одному файлі виділення блоків за допомогою **табуляцій та пробілів одночасно**.

Ви також можете виділяти декілька рівнів вкладеності, додаючи ще 4 пробіли зліва для всіх інструкцій блоку:

x = int(input("X: "))  
y = int(input("Y: "))  
  
if x == 0:  
 print("X can`t be equal to zero")  
 x = int(input("X: "))  
  
 if x == 0:  
 print("X can`t be equal to zero")  
 x = int(input("X: "))  
  
 if x == 0:  
 print("X can`t be equal to zero")  
 x = int(input("X: "))  
  
result = y / x

В цьому прикладі тричі повторюється перевірка на нерівність x нулю, і на кожну перевірку блок інструкцій виділяється додатковими 4-ма пробілами.

Приклад вкладеності для визначення чвертей для координатної площини.

if x >= 0:  
 if y >= 0: *# x > 0, y > 0*  
 print("Перша чверть")  
 else: *# x > 0, y < 0*  
 print("Четверта чверть")  
else:  
 if y >= 0: *# x < 0, y > 0*  
 print("Друга чверть")  
 else: *# x < 0, y < 0*  
 print("Третя чверть")

**Цикли**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%86%D0%B8%D0%BA%D0%BB%D0%B8)

Для того, щоб повторити якийсь блок коду кілька разів або повторювати, доки виконується деяка умова, у Python реалізовані цикли. Існує два види циклів:

* цикл for, який ще називають ітеруючим, він перебирає усі елементи деякої послідовності;
* цикл while, який виконується, доки виконується деяка умова.

**Ітерація (лат. iteratio «повторювання») — повторювання будь-якої дії.**Ітерація у програмуванні — організація обробки даних, за якої дії повторюються багаторазово, не призводячи, при цьому, до викликів самих себе.

Одна ітерація — це одне повторювання.

**Цикл for**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%86%D0%B8%D0%BA%D0%BB-for)

У Python цикл for використовується для перебору усіх елементів контейнерів або ітерованих об'єктів, наприклад, списків. Інструкції, які знаходяться у тілі циклу, будуть виконані стільки разів, скільки елементів у списку.

При цьому на кожній ітерації спеціальна змінна набуває значення одного з елементів списку.

Роботу циклу for можна порівняти з тим, що ви по черзі візьмете кожну літеру з фрази й промовите її. Фразою буде виступати рядок 'apple', а аналогом вимовлення вголос буде виступати виведення відповідної літери в консоль.

fruit = 'apple'  
for char in fruit:  
 print(char)

У результаті виконання цього коду ви побачите в консолі:

a  
p  
p  
l  
e

Синтаксис циклу for:

1. цикл розпочинається з ключового слова for;
2. за яким обов'язково йде назва змінної, куди записуватиметься значення, що отримується з ітерованого об'єкту на кожній ітерації;
3. далі слідує ключове слово in;
4. за яким обов'язково йде вираз або об'єкт, по якому, власне, буде ітеруватися for;
5. далі ставиться :;
6. і з нового рядка з відступом йде набір виразів, які повторюватимуться на кожній ітерації

**Цикл while**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D1%86%D0%B8%D0%BA%D0%BB-while)

Цикл while дозволяє виконати інструкції, які знаходяться у тілі циклу до тих пір, доки виконується умова, вказана в циклі. Наприклад, цикл while, який виводить числа від 1 до 5:

a = 1  
while a <= 5:  
 print(a)  
 a = a + 1

**«Нескінченні цикли» та break**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%BD%D0%B5%D1%81%D0%BA%D1%96%D0%BD%D1%87%D0%B5%D0%BD%D0%BD%D1%96-%D1%86%D0%B8%D0%BA%D0%BB%D0%B8-%D1%82%D0%B0-break)

Бувають ситуації, коли необхідно вийти з циклу до завершення ітерації, не дочекавшись, доки станеться чергова перевірка умови. Для цього є команда break. Команда break зупиняє цикл в момент виклику і не завершує ітерацію.

a = 0  
while True:  
 print(a)  
 if a >= 20:  
 break  
 a = a + 1

В цьому прикладі умова циклу буде виконуватися завжди, адже True завжди буде True. Це приклад нескінченного циклу. Але через перевірку, що a >= 20, цей цикл завершиться, щойно в a буде значення 20 або більше.

Нескінченні цикли часто застосовуються там, де потрібно взаємодіяти з клієнтом, чекаючи введення від нього, і завершується тільки при настанні деякої умови.

Наприклад echo скрипт, який виводить в консоль те, що ви введете, доки ви не введете exit:

while True:  
 user\_input = input()  
 print(user\_input)  
 if user\_input == "exit":  
 break

**Завершення ітерації за допомогою continue**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%B7%D0%B0%D0%B2%D0%B5%D1%80%D1%88%D0%B5%D0%BD%D0%BD%D1%8F-%D1%96%D1%82%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97-%D0%B7%D0%B0-%D0%B4%D0%BE%D0%BF%D0%BE%D0%BC%D0%BE%D0%B3%D0%BE%D1%8E-continue)

Також для того, аби одразу перейти до наступної ітерації циклу без виконання виразів, що залишилися, є команда continue. Виклик цієї команди у тілі циклу призводить до того, що вирази цієї ітерації, що залишилися, не будуть виконані, а інтерпретатор одразу перейде до наступної ітерації або перевірки умови.

a = 0  
while a < 6:  
 a = a + 1  
 if not a % 2:  
 continue  
 print(a)

У консолі ви побачите:

1  
3  
5

Інструкція print(a) не виконувалась, коли a ділилося на 2 без залишку, оскільки ітерація завершувалася за допомогою continue.

В цьому прикладі використовувався оператор отримання залишку від ділення %, він повертає таке число p, що якщо його відняти від r, то результат буде ділитися на x націло:

(r - p) / x = a,

де а, x, r — цілі числа.

Оператори continue та break працюють тільки всередині одного циклу. В ситуації вкладених циклів немає способу вийти з усіх циклів одразу.

while True:  
 number = input("number = ")  
 number = int(number)  
 while True:  
 print(number)  
 number = number - 1  
 if number < 0:  
 break

В цьому прикладі користувач вводить число та отримує зворотний відлік від цього числа до 0 в консолі. При цьому, зовнішній нескінченний цикл жодним чином не перервати і break вийде тільки з внутрішнього циклу.

Також використання continue або break поза циклом призводить до синтаксичної помилки.

number = int(input("number = "))  
if number < 0:  
 break

Такий код призводить до помилки SyntaxError. Такі помилки називаються виключеннями.

**Винятки**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D0%B8)

Перетворити в int або float можна не будь-який рядок. Наприклад, якщо користувач введе 'a', то інтерпретатор не зможе визначити, як перетворити символ a в ціле число, і викличе виняток ValueError.

int("a")  
---------------------------------------------------------------------------  
ValueError Traceback (most recent call last)  
<ipython-input-6-d9136db7b558> in <module>  
----> 1 int("a")  
  
ValueError: invalid literal for int() with base 10: 'a'

Виключення у Python — це помилка на рівні інтерпретатора, викликана неможливістю виконати той або інший оператор з будь-яких причин (змінна не існує, синтаксична помилка, відсутній атрибут, операція ділення на нуль тощо).

У нашому прикладі (ввели 'а') інтерпретатор намагається перетворити рядок в тип int (ціле число), але як перетворити рядок 'a' у число не визначено і буде викликаний виняток із цього приводу.

**Механізм обробки винятків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%BC%D0%B5%D1%85%D0%B0%D0%BD%D1%96%D0%B7%D0%BC-%D0%BE%D0%B1%D1%80%D0%BE%D0%B1%D0%BA%D0%B8-%D0%B2%D0%B8%D0%BD%D1%8F%D1%82%D0%BA%D1%96%D0%B2)

Для обробки винятків існує оператор try ... except .... Синтаксично, цей оператор розпочинається з ключового слова try: (спробувати) та продовжується блоком коду, в якому ми чекаємо, що може статися помилка.

Далі йде блок обробки винятків except (крім), де можна вказати один або більше винятків. Якщо один із зазначених винятків станеться, то виконається наступний блок коду.

Цей блок не обов'язковий, але найчастіше потрібний. Він виконається, якщо станеться зазначений виняток (один із них, якщо їх декілька).

Якщо помилки можуть бути різні і обробляти їх потрібно теж по-різному, то можна додати декілька блоків except, у кожному вказати свою помилку і що робити, якщо вона станеться.

Потім йде необов'язковий блок, який розпочинається з ключового слова else. Цей код виконається, тільки якщо винятків не сталося.

Останнім йде необов'язковий блок коду, який розпочинається з ключового слова finally, він виконається у будь-якому разі, незалежно від того, були помилки або ні.

У нашому прикладі обробка призначеного для користувача введення виглядатиме таким чином:

val = 'a'  
try:  
 val = int(val)  
except ValueError:  
 print(f"val {val} is not a number")  
else:  
 print(val > 0)  
finally:  
 print("This will be printed anyway")

Винятки у Python — це дуже потужний інструмент, який часто використовується для управління потоком виконання, а не тільки для обробки помилок. У динамічних мовах ніколи не можна бути на 100% впевненим у тому, що користувач ввів значення коректного типу або, що інший застосунок не повернув None, замість int, наприклад.

Наївним розв'язання цієї проблеми буде повсюдне використання перевірок if на коректність введеного користувачем або іншим застосунком значення. Просунутішим, зручнішим і прозорішим рішенням є використання механізму обробки винятків там, де вони можуть статися через некоректні вхідні дані.

age = input("How old are you? ")  
try:  
 age = int(age)  
 if age >= 18:  
 print("You are adult.")  
 else:  
 print("You are infant")  
except ValueError:  
 print(f"{age} is not a number")

**Основні типи виключень у Python**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson02/lesson-02/#%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D1%82%D0%B8%D0%BF%D0%B8-%D0%B2%D0%B8%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%8C-%D1%83-python)

SyntaxError — синтаксична помилка.

IndentationError — помилка, яка виникає, якщо у виділенні блоків інструкцій пробілами припущена помилка.

TabError виникає, якщо в одному файлі використовувати пробіли і табуляції для виділення блоків інструкцій.

TypeError виникає, коли операція зі змінною цього типу неможлива.

2 / 'a'

ValueError виникає, коли тип операнда відповідний, але значення таке, що операцію неможливо виконати.

int("a")

ZeroDivisionError — ділення на нуль.