Миколаївський обласний інститут
післядипломної педагогічної освіти
Кафедра теорії й методики природничо-
математичної освіти та інформаційних технологій

Н. А. Махровська, Г. С. Погромська

Старт у світ кодування: Python для початківців

Миколаїв
2026

СЗ “Старт у світ кодування: Python для початківців”

Рекомендовано до друку рішенням вченої ради Миколаївського обласного інституту післядипломної педагогічної освіти, протокол від 25 грудня 2025 року № 7.

Автори: ***Н. А. Махровська**, кандидат фізико-математичних наук, доцент кафедри теорії й методики природничо-математичної освіти та інформаційних технологій МОІППО;*

***Г. С. Погромська,** кандидат педагогічних наук, доцент, доцент кафедри теорії й методики природничо-математичної освіти та інформаційних технологій МОІППО*

Рецензенти: ***С. О. Слободян**, кандидат технічних наук, професор, проректор з науково-педагогічної роботи Національного університету кораблебудування імені адмірала Макарова, м. Миколаїв;*

***С. І. Ємельянов,** доктор філософії з фізики та астрономії, старший викладач кафедри економічної кібернетики, комп’ютерних наук та інформаційних технологій Миколаївського національного аграрного університету*

Відпровідальна за випуск: ***В. В. Стойкова,** кандидат педагогічних наук, заступник директора з науково-педагогічної роботи МОІППО*

У методичному посібнику представлений матеріал з вивчення основ мови програмування Python та прикладів програм, що ґрунтуються на базових алгоритмічних структурах.

Махровська Н. А., Погромська Г. С. Старт у світ кодування: Python для початківців. – Миколаїв : центр редакційно-видавничої діяльності МОІППО, 2026. – 238 с.

© Кафедра теорії й методики природничо-математичної освіти та ІТ МОІППО
© Центр редакційно-видавничої діяльності МОІППО
2026

ЗМІСТ

1. Вступ. IDE для Python. Введення-виведення даних
2. Змінні, типи даних, основні операції мови Python
3. Цілочислова арифметика
4. БАС: структура слідування
5. Логічні операції
6. Структура розгалуження
7. Структура вибору
8. Структура повторення: цикл for
9. Цикл for: функція range
10. Структура повторення: цикл while
11. Оператори переривання
12. Вкладені цикли
13. Рядковий тип даних
14. Основи роботи зі списками

ПЕРЕДМОВА

Сучасний світ усе більше залежить від технологій, і базові знання програмування стають такою ж важливою навичкою, як читання або письмо. Розуміння основ кодування покращує адаптацію до змін у суспільстві, в якому домінують цифрові інструменти. Процес програмування покращує розвиток аналітичного мислення, розв’язання задач і креативності. Знання програмування корисне не лише в IT-сфері, а й таких галузях, як біологія, медицина, архітектура, дизайн, інженерія тощо.

Вивчення Python у школі є актуальним через його універсальність, простоту, відповідність сучасним освітнім і професійним вимогам. Python – мова програмування для початківців та водночас потужний інструмент професійних розробників.

ПЕРЕДМОВА

За новими програмами початкової школи, адаптаційного (5–6 класи) та базового предметного навчання (7–9 класи) програмування вивчається з різним ступенем заглиблення у фактичний матеріал відповідно до вікової категорії здобувачів освіти. У мові програмування Python підтримується декілька парадигм програмування: об'єктно-орієнтована, процедурна, функціональна, візуальна. Використання мови програмування Python в освітньому процесі сприятиме ґрунтовному засвоєнню основних алгоритмічних структур та професійних прийомів програмування.

ПЕРЕДМОВА

Навчально-методичний посібник містить матеріали до однойменного авторського курсу «Старт у світ кодування: Python для початківців». Зміст доступний за швидкокодом:

Пропонований матеріал розрахований на вчителів інформатики, вчителів природничо-математичного циклу, які інтегрують програмування у свої навчальні дисципліни, та всіх зацікавлених, хто має на меті вдосконалити та систематизувати знання основ мови програмування Python та навички розв’язування задач, що ґрунтуються на базових алгоритмічних структурах.

НАВЧИМОСЯ:

- встановлювати та запускати інтерпретатор Python;
- працювати з даними;
- працювати з числовими типами даних **int** і **float;**
- працювати з умовним оператором **if-elif-else;**
- застосовувати цикли **for** і **while;**
- обробляти рядковий тип даних **str**



ЗАНЯТТЯ 1.

ВСТУП. IDE ДЛЯ PYTHON. ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ



ВСТУП ДО ПРОГРАМУВАННЯ

КОМП’ЮТЕРНА ПРОГРАМА

... список команд (інструкцій) для комп'ютера, написаних мовою програмування.



СКЛАД МОВИ ПРОГРАМУВАННЯ

Алфавіт мови	Синтаксис мови	Семантика мови
Набір символів, що може використовуватись у програмному коді	Сукупність правил побудови команд мовою програмування	Сукупніть правил виконання комп’ютерних команд, записаних мовою програмування

МОВА ПРОГРАМУВАННЯ ТА ЇЇ СКЛАД

1

МОВА ПРОГРАМУВАННЯ

… набір певних правил, за якими комп'ютер може розуміти команди (інструкції) і виконувати їх.

2

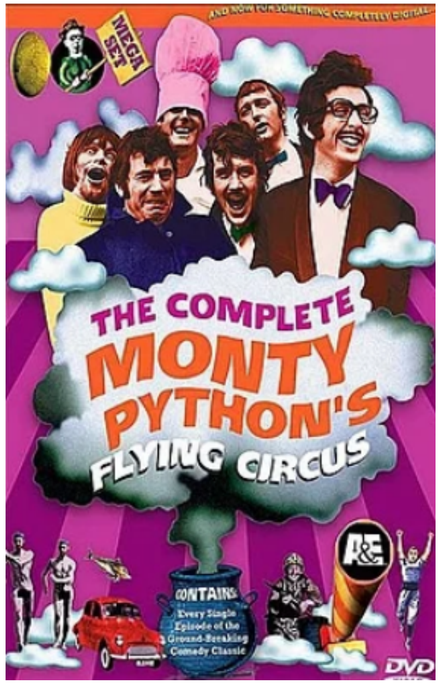
Текст програми будь-якою мовою програмування називається **програмним кодом або програмою**



МОВА ПРОГРАМУВАННЯ PYTHON

Мову Python розробив голландський програміст Гвідо Ван Россум (Guido van Rossum) 1991 року. Гвідо був фанатом британського комедійного серіалу «Monty Python's Flying Circus», звідки й пішла назва мови.





ПЕРЕВАГИ І НЕДОЛІКИ МОВИ PYTHON



- інтерпретується
- платформо-незалежна
- проста
- вбудована скриптова
- динамічна типізація
- велика бібліотека класів



- низька швидкість виконання порівняно з C і C++
- динамічна типізація мови - мінус при написанні складних програмних комплексів



ЗАВДАННЯ, ЩО ВИРІШУЮТЬСЯ ЗА ДОПОМОГОЮ PYTHON:

- СИСТЕМНЕ ПРОГРАМУВАННЯ
- ГРАФІЧНІ ДОДАТКИ
- ВЕБ-ДОДАТКИ
- ВЕБ-СЦЕНАРІЇ
- ІНТЕГРАЦІЯ КОМПОНЕНТІВ
- ДОДАТКИ БАЗ ДАНИХ

ПРОЄКТИ, В ЯКИХ ВИКОРИСТОВУЄТЬСЯ PYTHON















СЕРЕДОВИЩА ПРОГРАМУВАННЯ

IDE - INTEGRAL DEVELOPMENT ENVIRONMENT

BEST CODE EDITORS FOR PYTHON

Idle

VSCode

PyCharm

ideone.com

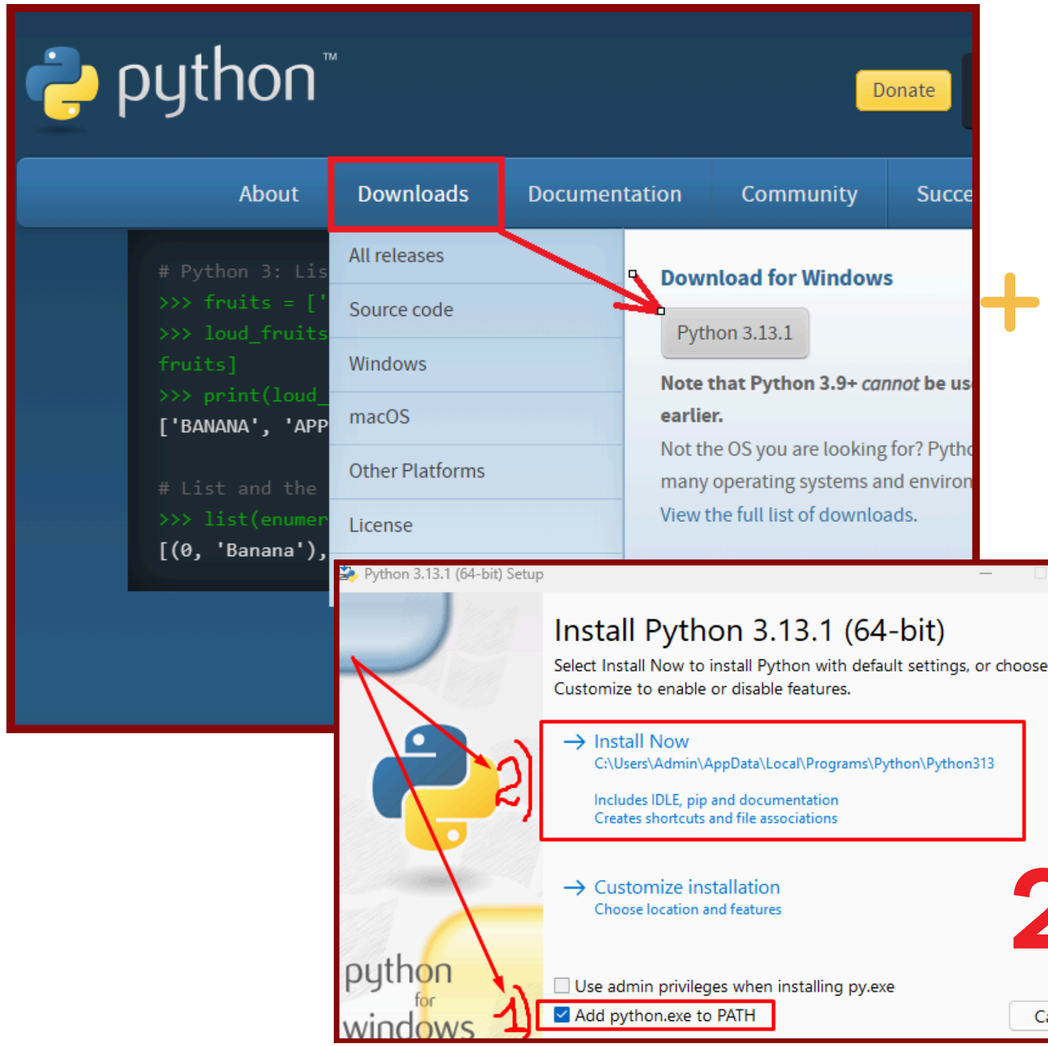
PyDroid3



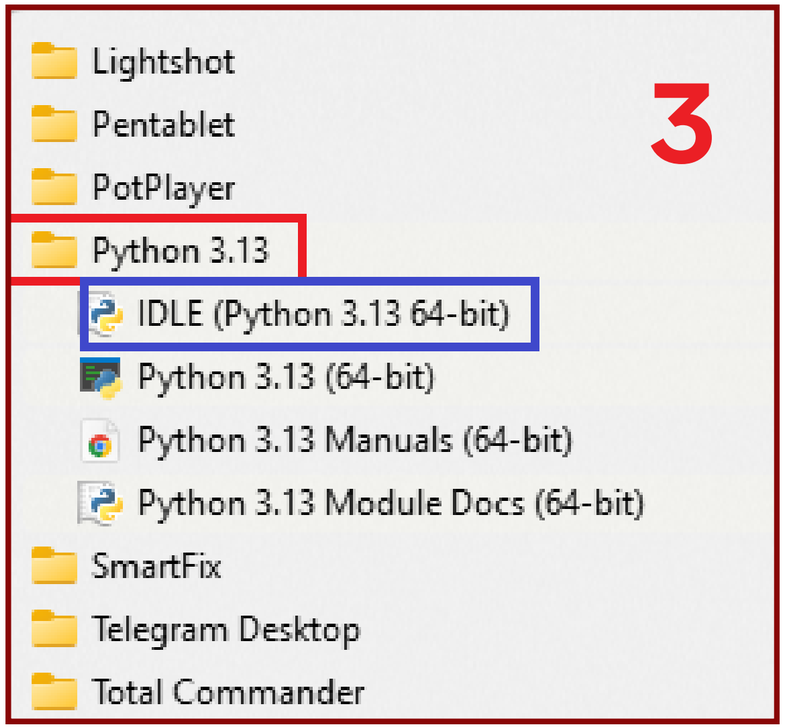
КРОК 1: IDLE

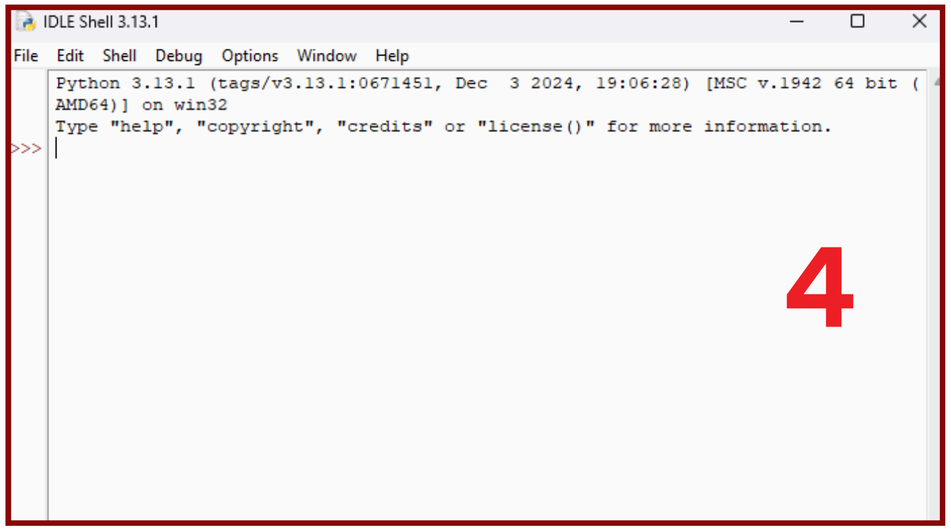
python.org

1



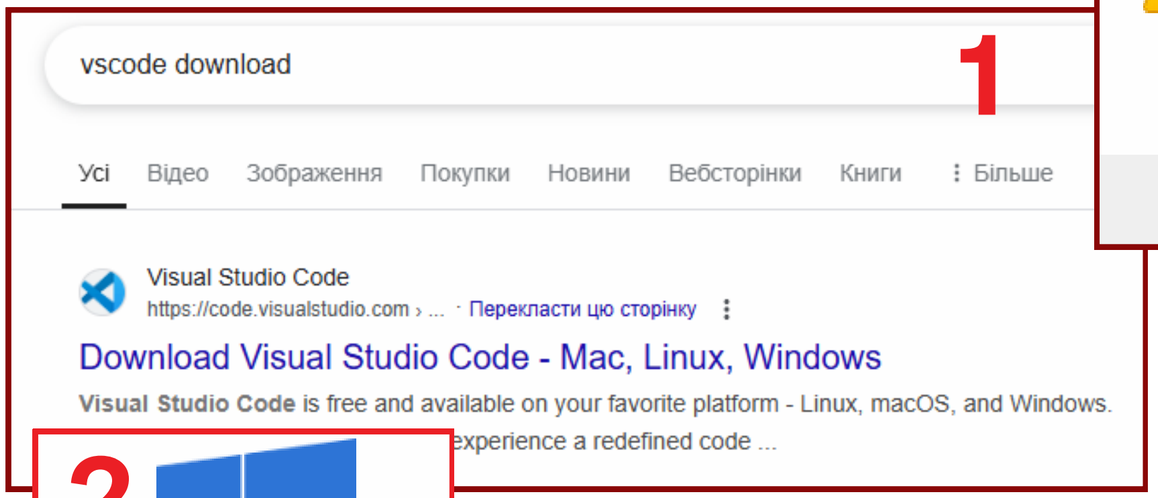
+

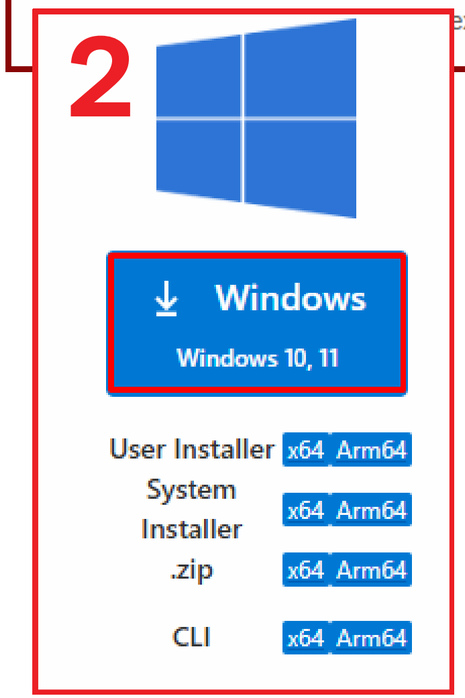


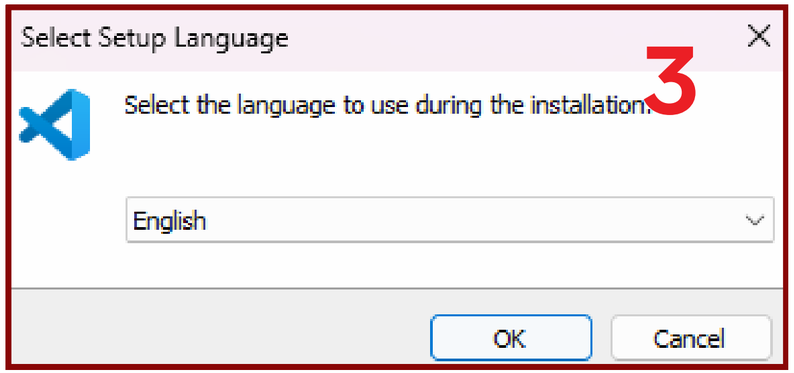


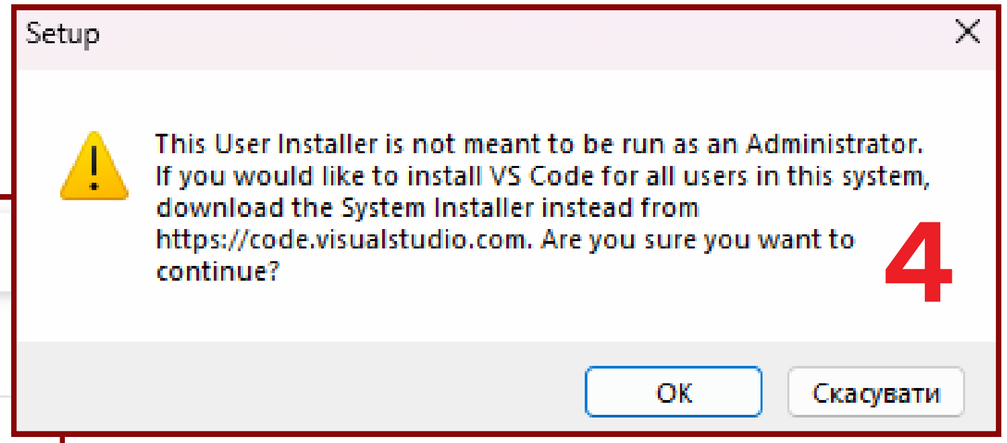
КРОК 2: VSCODE

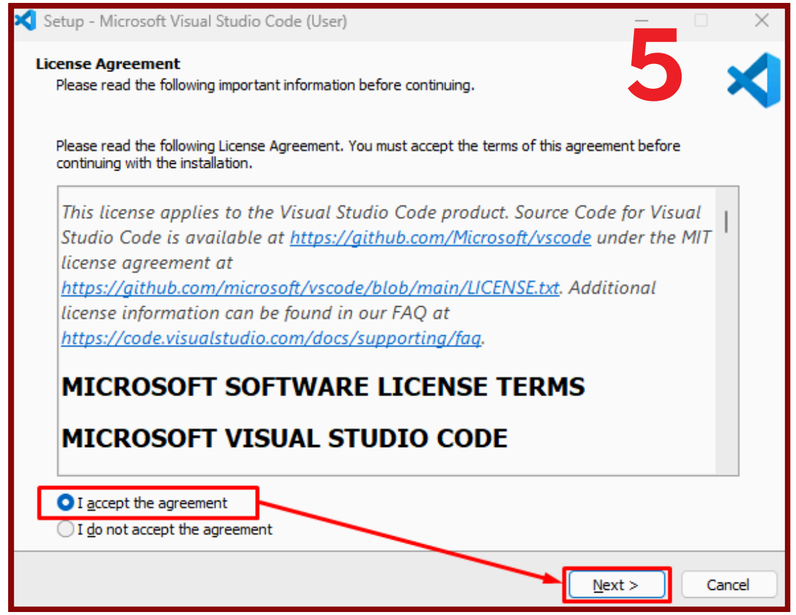
VSCode download





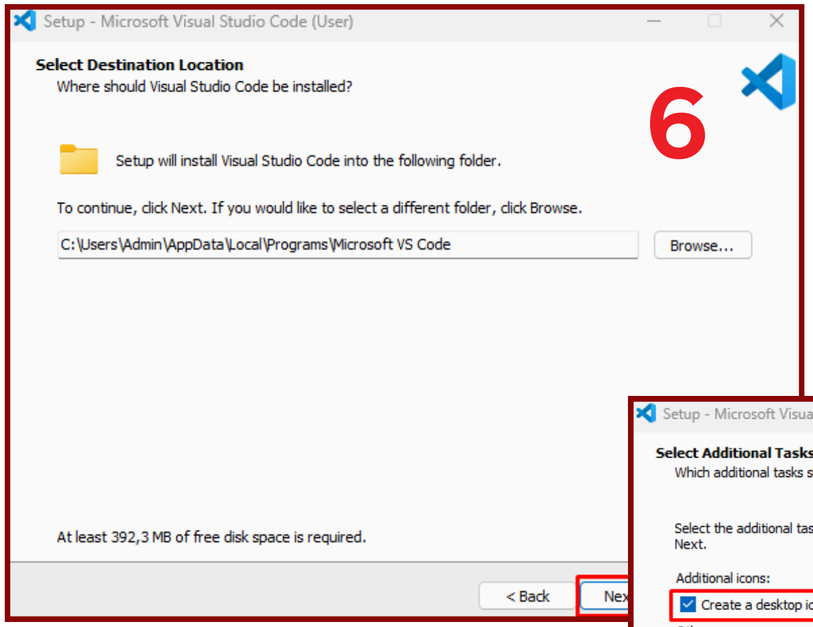


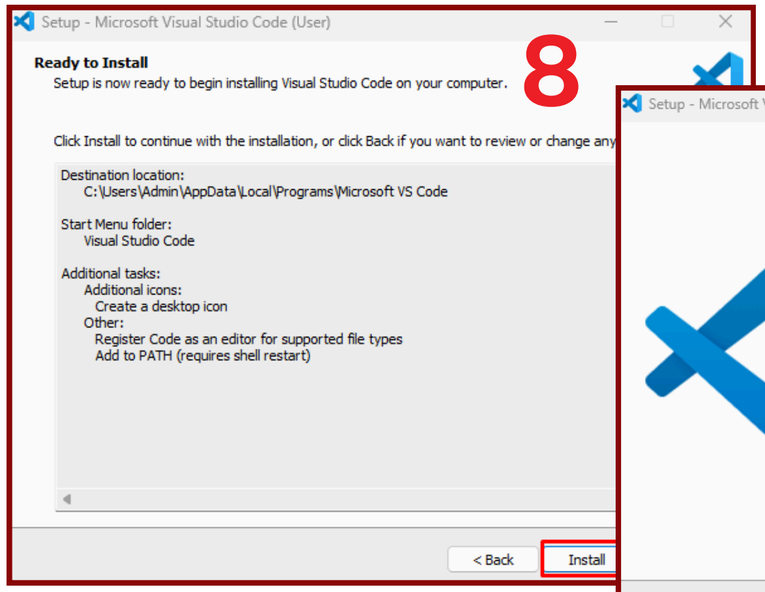


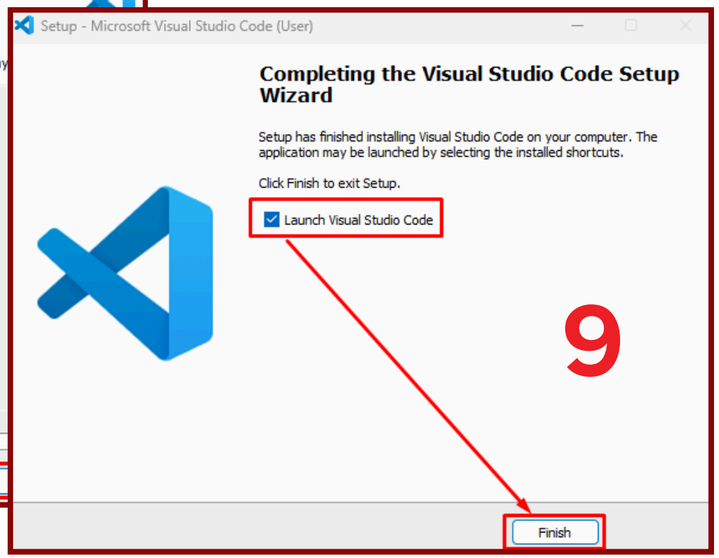


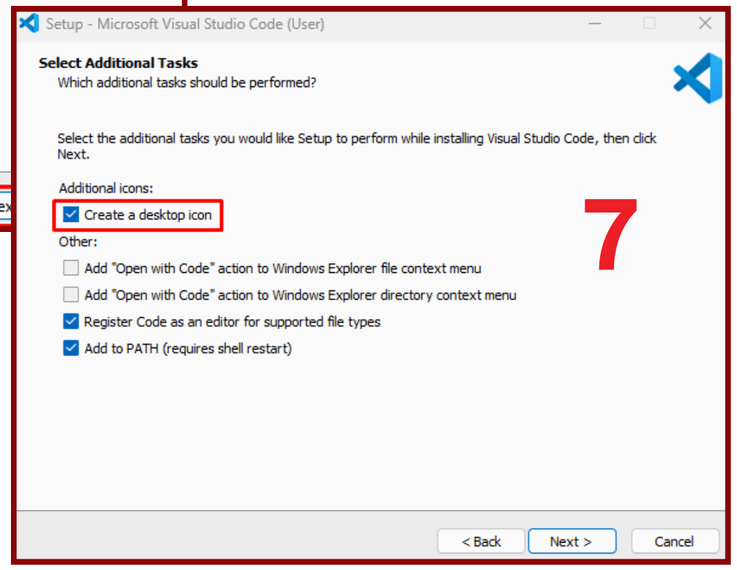
KPOK 2: VSCODE

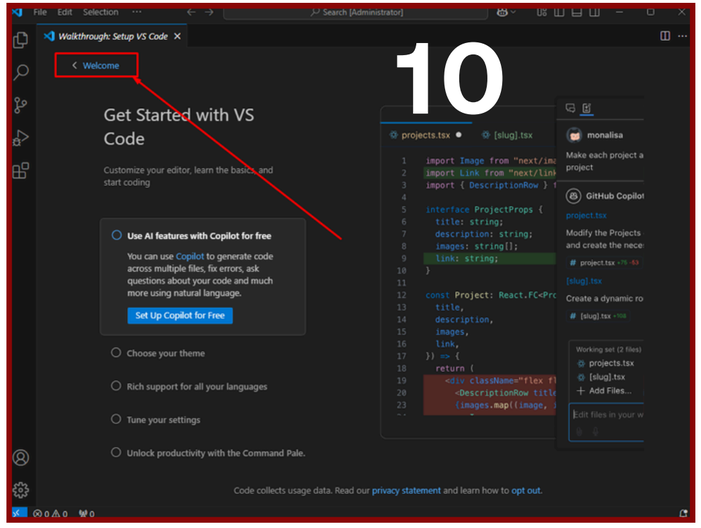
VSCode download











THE ZEN OF PYTHON

```
import this
```

- Beautiful is better than ugly;
- Explicit is better than implicit;
- Simple is better than complex;
- Complex is better than complicated;
- Flat is better than nested;
- Sparse is better than dense;
- Readability counts;
- Special cases aren't special enough to break the rules



...

IDLE



- Красиве краще, ніж потворне;
- Явне краще, ніж неявне;
- Просте краще, ніж складне;
- Складне краще, ніж заплутане;
- Плоске краще, ніж вкладене;
- Розріджений краще, ніж щільний;
- Читабельність має значення;
- Особливі випадки не настільки особливі, щоб порушувати правила

...

ОСНОВНІ УЗГОДЖЕННЯ МОВИ PYTHON

```
import this
# Приклад програми з відступами
print('Hello, Python')
if 5 > 8:
    print('Це не наша математика')
else:
    print("It's OK")
print("The END!")
```

Python використовує відступ для позначення блоку коду (зазвичай 4 пробіли)

Однорядкові коментарі починаються з символу #

Мова Python не має синтаксису для багаторядкових коментарів
спосіб написання багаторядкових коментарів — потрійні лапки ''' або """

Інтерпретатор ігнорує рядок, який не присвоєно жодній змінній або функції

КОМАНДА ВИВЕДЕННЯ ДАНИХ PRINT()

```
print(1, 2, 3, sep='__', end='   ')
print(4, 5, sep="***", end=':::')
print(6, sep='----')
```

```
print('Hello!')
print("Happy Pythoning")
print('Everything', 'will', 'be', "GOOD")
```

Функція print() виводить вказаний у дужках об’єкт на стандартний пристрій виводу (екран) або у файл **текстового потоку.**

Усередині круглих дужок пишемо те, що слід вивести на екран.
Якщо це **текст**, обов'язково вказуємо його всередині **лапок**.

Лапки можуть бути одинарними (') або подвійними (")

```
print('Python')
print("Python")
```

Аргументи - це конкретні значення, які передаються функції під час її виклику.

Команда *print()* дозволяє вказувати **кілька аргументів**, в такому разі їх треба відокремлювати **комами**.

- **Зверніть увагу**, як роздільник під час виведення даних між аргументами команди використовується символ пробілу.
- За замовчуванням команда print() додає рівно один пробіл між усіма своїми аргументами.

```
print('Я', 'вчусь', 'програмувати','на','python')
```

```
Я вчусь програмувати на python
```

При написанні коду **між аргументами** команди print() після коми ставимо один символ **пропуску**. Це загальноприйнята угода у мові Python. Цей символ пропуску не впливає на виведення даних. Це прийнято для читабельності коду.



Примітка 1. Команда print() записується маленькими літерами, інше написання неприпустиме, в Python *малі та великі літери інтерпретуються по різному*.

Примітка 2. Кожна наступна команда print() виводить текст-аргумент із нового рядка.

Примітка 3. Команда print() з порожнім списком аргументів вставляє новий порожній рядок.

Примітка 4. В Python можна використовувати як одинарні, так і подвійні лапки для обрамлення тексту. Оскільки часто лапки є частиною тексту, то для правильного розпізнавання такого тексту, користуємося різними лапками:

- якщо в тексті є одинарні лапки, то для обрамлення такого тексту використовуємо подвійні лапки;
- якщо в тексті є подвійні лапки, то обрамляємо його одинарними лапками.

Примітка 5. У одному print() можна комбінувати одинарні та подвійні лапки. Це робить код гнучкішим, даючи змогу легко включати лапки й апострофи в рядки.

Примітка 6. Не можна використовувати одинарні лапки в рядку, якщо сам рядок обрамлений одинарними лапками. З подвійними лапками ситуація аналогічна.

ПАРАМЕТРИ КОМАНДИ ВИВЕДЕННЯ



За замовчуванням команда print() приймає **кілька аргументів**, потім виводить їх через пробіл, після чого робить переведення курсора на наступний рядок. Таку поведінку можна змінити, використовуючи ***необов'язкові*** параметри **sep** і **end**.

```
a = 2
b = 3
print(a, '+', b, '=', a + b, sep="")
```

!!! без пробілів за рахунок sep

```
2+3=5
```

Параметр **sep** (separator – роздільник) команди print() дозволяє встановити рядок, за допомогою якого будуть розділені аргументи (якщо їх декілька) під час друку. *За замовчуванням* цей параметр дорівнює *символу пропуску*.

Параметр **end** визначає, що потрібно додати до кінця виводу. *За замовчуванням* параметр end дорівнює *символу перекладу рядка (\n)*.

Параметр end визначає рядок, який буде додано після виведення всіх аргументів команди print().



Примітка 7. Виклик команди print() з порожніми дужками робить переведення курсору на новий рядок.

Примітка 8. Послідовність символів називається керуючою послідовністю і задає переклад рядка.

Примітка 9. Параметри *sep* та *end* можна встановлювати одночасно.

```
print('a', 'b', 'c', sep='*', end='finish')
```

Примітка 10. Для різних команд print() можна задавати різні параметри sep та end.

Примітка 11. Щоб прибрати всі додаткові символи, можна встановити параметри *sep* і *end* команди print() як порожні рядки (''').

Примітка 12. Якщо після виведення даних потрібно більше одного перекладу рядка, можна використовувати наступний код:

```
print('Python', end='\n\n\n')
```

Примітка 13. Параметр *sep* є роздільником кількох аргументів у команді print(). Якщо аргумент у команді print() лише один, то параметру *sep* нема чого розділяти. У такому разі параметр *sep* ніяк не впливатиме на текст, що виводиться..

КОМАНДА ВВЕДЕННЯ ДАНИХ INPUT()

Програми можуть зчитувати дані, а потім їх використовувати.

Команда input() зчитує дані, введені з клавіатури.

```
a = input()
b = input()
c = a + b
print (c)
```

ввести рядок з клавіатури і зв'язати зі змінною a

Протокол:

```
21
33
2133
```

? Чому?

! Результат функції **input** – рядок символів!

перетворити в ціле число

```
a = int(input())
b = int(input())
```

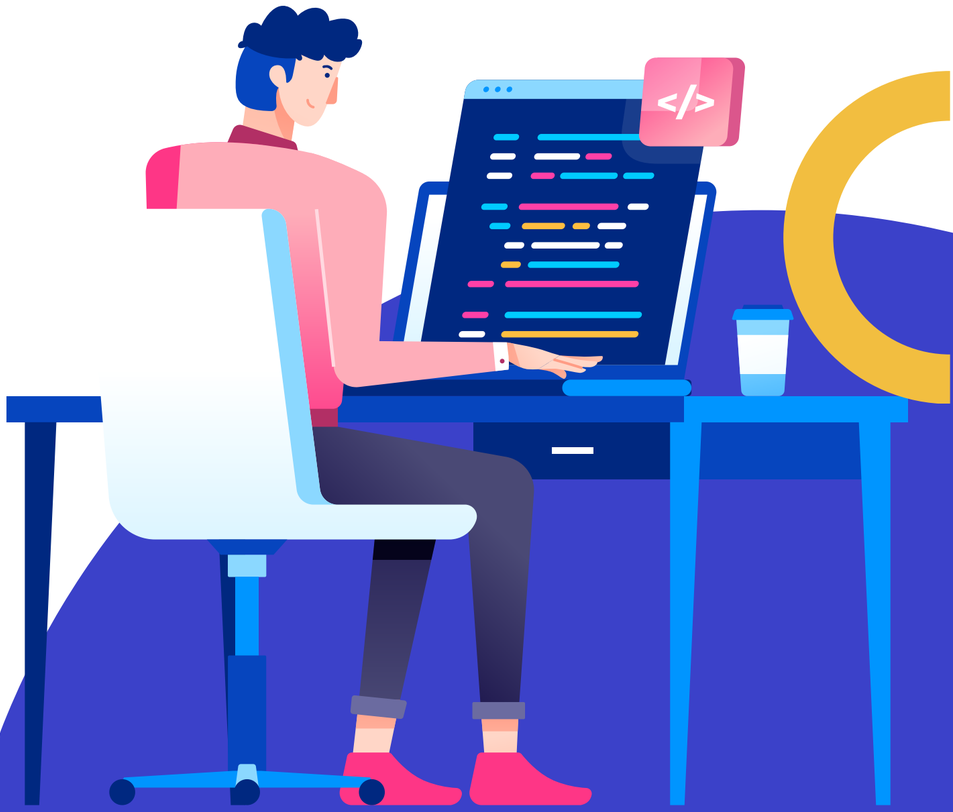
Принцип роботи: коли програма “зустрічає” команду input(), вона чекає, доки користувач введе текст із клавіатури (введення завершується натисканням клавіші Enter). Введений рядок підставляється на місце input()

```
print("Як твоє ім'я?")
print('Привіт,', input())
```

```
Як твоє ім'я?
Анна
Привіт, Анна
```

ЗАНЯТТЯ 2.

«ЗМІННІ, ТИПИ ДАНИХ, ОСНОВНІ ОПЕРАЦІЇ МОВИ PYTHON»



1. **Комп'ютерна програма - ...**

2. **Для чого використовується команда print()?**

3. **Яка дія команди input()?**

4. **Параметр sep ...**

Обери правильний варіант

алгоритм	набір команд	написана мовою програмування	
введення даних		виведення даних	
створює комірку	очікує на введення даних	вводить дані	
обов’язковий	необов’язковий	додає пробіл	додає авторський знак

ЗМІННІ

Змінна - це іменована ділянка пам'яті, в якій зберігаються дані.

```
name = 'python'
ball = input()
print('Я знаю', name, 'на', ball)
```

Будь-яка змінна має ім'я і значення

*у змінній **name** зберігається рядок **'python'***
*у змінній **ball** зберігається текст, що вводиться користувачем*

```
10
Я знаю python на 10
```

ІМ’Я ЗМІННОЇ

… це послідовність латинських літер та цифр, що починається з букви.
Символ **“_”** (підкреслення) теж вважається літерою.

Правила створення імен змінних:

- може містити тільки латинські літери a-z, A-Z, цифри та символ нижнього підкреслення _
- не може починатися з цифри
- не може містити пробіли
- по можливості має відображати її призначення

Правильні	Неправильні
name1 my_variable _sum a TotalAmount qwerty12my	5name змінна surname$%

ЗАВДАННЯ

Які з імен допустимі для назв змінних у Python?

1. teacher_2
2. teacher 2
3. 2teacher
4. teacher2
5. teacher
6. 2 teacher

РЕГІСТРОЗАЛЕЖНІСТЬ

- **name** і **Name** - дві абсолютно різні змінні.

- Для іменування змінних прийнято використовувати стиль **lower_case_with_underscores** (маленькі літери з підкресленнями).
- *Значення змінної* - інформація, що зберігається у відповідній комірці пам’яті змінної.
- У змінній може зберігатися текст, число тощо.

СТВОРЕННЯ ЗМІННОЇ

Ім'я змінної завжди має бути ліворуч від знака рівності

Створити змінну: `<ім’я_змінної> = <значення_змінної>`

оператор присвоєння

Значення змінної.

- Значення змінної можна переприсвоювати.
- Оператор присвоювання надає змінній значення незалежно від того, була ця змінна введена (оголошена) раніше чи ні.
- Якщо вже оголошена деяка змінна, можна використовувати її значенням за потребою.

ПРИКЛАДИ РОБОТИ ЗІ ЗМІННИМИ

```
disc = 'Math'
disc = 'Informatics'
print(disc)
```

```
Informatics
```

```
zm = 12
inzm = zm
print(inzm)
```

Яке значення буде міститися у змінній s1 після виконання наступного коду?

```
s1 = 'C++'
s2 = 'Python'
s3 = 'Java'
s3 = s2
s1 = s3
```

PEP 8

… це документ, що описує загальноприйнятий стиль написання програм мовою Python.
Документ створено за рекомендаціями Гвідо Ван Россума

PEP = Python Enhancement Proposal

РЕКОМЕНДАЦІЇ PEP 8

1

Рекомендація 1. Уникайте використання пробілів одразу перед відкритою дужкою, після якої починається список аргументів функції.

Правильно:

```
print('Follow PEP8!')
```

Неправильно:

```
print ('Follow PEP8!')
```

РЕКОМЕНДАЦІЇ PEP 8

2

Рекомендація 2. Після коми обов'язково потрібно додавати рівно один пробіл.

Правильно:

```
print('PEP8!', 'Rocks!')
```

Неправильно:

```
print('PEP8!','Rocks!')
```

```
print('PEP8!',      'Rocks!')
```

РЕКОМЕНДАЦІЇ PEP 8

3

Рекомендація 3. Не використовуйте пробіли для відокремлення знака =, коли він вживається для позначення необов'язкових параметрів

Правильно:

```
print('My name', 'in', 'Python', sep='**', end='+')
```

Неправильно:

```
print('My name', 'in', 'Python', sep = '**', end = '+')
```

ЦІЛІ ЧИСЛА

- Усі попередні програми, працювали з **текстовими даними,** але частіше виникає потреба працювати саме з **числами.**
- Для того, щоб у мові Python створити змінну цілого типу, не потрібно брати число в лапки при оголошенні змінної

```
num1 = 7
num2 = 10
num3 = num1 + num2
print(num3)
```

Буде виведено число 17

Число записується без лапок, а рядки в лапках

АРИФМЕТИЧНІ ОПЕРАЦІЇ. ПРІОРИТЕТ

Для зміни пріоритету потрібні дужки

У Python над числами можна здійснювати **4 основні арифметичні операції.** Знак операції ставиться **між** операндами:

Операція	Опис	Пріоритет
+	додавання	2
-	віднімання	2
*	множення	1
/	ділення	1

```
a = 3
b = 2
print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

```
5
1
6
1.5
```

```
num1 = 2 + 3 * 4
num2 = (2 + 3) * 4
print(num1)
print(num2)
```

```
14
20
```

ПЕРЕТВОРЕННЯ РЯДКА НА ЦІЛЕ ЧИСЛО

Щоб перетворити **рядок** на **ціле число**, використовуємо команду **int():**

- Змінна *age* має рядковий тип
- Змінна *year* має цілий тип
- Змінна *grade* має цілий тип

```
age = '1992'
year = int(age)
grade = int(input())
```

int(): візьми те, що зазначено в дужках і перетвори це на ціле число

ПЕРЕТВОРЕННЯ РЯДКА НА ЦІЛЕ ЧИСЛО

Приклад програми, яка зчитує два цілих числа і виводить на екран їх суму:

```
num1 = int(input())
num2 = int(input())
print(num1 + num2)
```

Щоб зчитати ОДНЕ ціле число, пишемо код:

```
num = int(input())
```

Скорочення **int (**від англ. **integer – цілий)**

У мові Python реалізована довга арифметика, завдяки якій ціла змінна обмежень не має

```
num = 10000000000000000000000000000000000000
```

ВВЕДЕННЯ ДАНИХ

Результат функції **input** – рядок символів!

перетворити в ціле число

```
a = int(input())
b = int(input())
```

Для зчитування ДВОХ (декількох) цілих чисел, заданих в одному рядку, пишемо код:

```
a, b = input().split()
a = int(a)
b = int(b)
```

або

```
a, b = map(int, input().split())
```

МНОЖИННЕ ПРИСВОЮВАННЯ

У мові Python за одну інструкцію присвоювання можна присвоїти значення відразу кільком змінним.

```
zm1, zm2 = 'Україна', 'Миколаїв'
print('змінна1 =', zm1, '-- змінна2=', zm2)
```



Цей код можна записати і так:

```
zm1 = 'Україна'
zm2 = 'Миколаїв'
print('змінна1 =', zm1, '-- змінна2=', zm2)
```

Відмінність двох способів полягає в тому, що множинне присвоювання в першому способі присвоює значення двом змінним одночасно.

Якщо потрібно зчитати текст із клавіатури і присвоїти його як значення змінним, це можна зробити так:

```
zm1, zm2 = input(), input()
print('змінна1 =', zm1, '-- змінна2=', zm2)
```

МНОЖИННЕ ПРИСВОЮВАННЯ

Якщо ліворуч від **знака присвоювання (=)** у множинному присвоюванні мають стояти через кому імена змінних, то праворуч можуть стояти довільні вирази, розділені комами. Головне, щоб ліворуч і праворуч від знака присвоювання було **однакове** число елементів.

Множинне присвоювання зручно використовувати, коли потрібно обміняти значення двох змінних. У мові Python це можна зробити так:

```
zm1, zm2 = 'Україна', 'Польща'
zm2, zm1 = zm1, zm2
print('zm1 =', zm1, '-- zm2 =', zm2)
```

```
zm1 = Польща -- zm2 = Україна
```

МНОЖИННЕ ПРИСВОЮВАННЯ

Зверніть увагу, що для обміну значень змінних наступний варіант не спрацює:

```
zm1, zm2 = 'Україна', 'Польща'
zm2 = zm1
zm1 = zm2
print('zm1 =', zm1, '-- zm2 =', zm2)
```

```
zm1 = Україна -- zm2 = Україна
```



Інструкція **zm2 = zm1** повністю заміняє попереднє значення змінної **zm2** (набуває значення 'Україна'). У інструкції **zm1 = zm2** присвоюємо змінній **zm2** значення змінної **zm1** (яке наразі є 'Україна')

ПРИМІТКИ

1. Назви змінних нічого не говорять інтерпретатору про значення в цих змінних, і навіть у дуже добре названій змінній не з'явиться потрібне значення, якщо його туди не записати у коді.

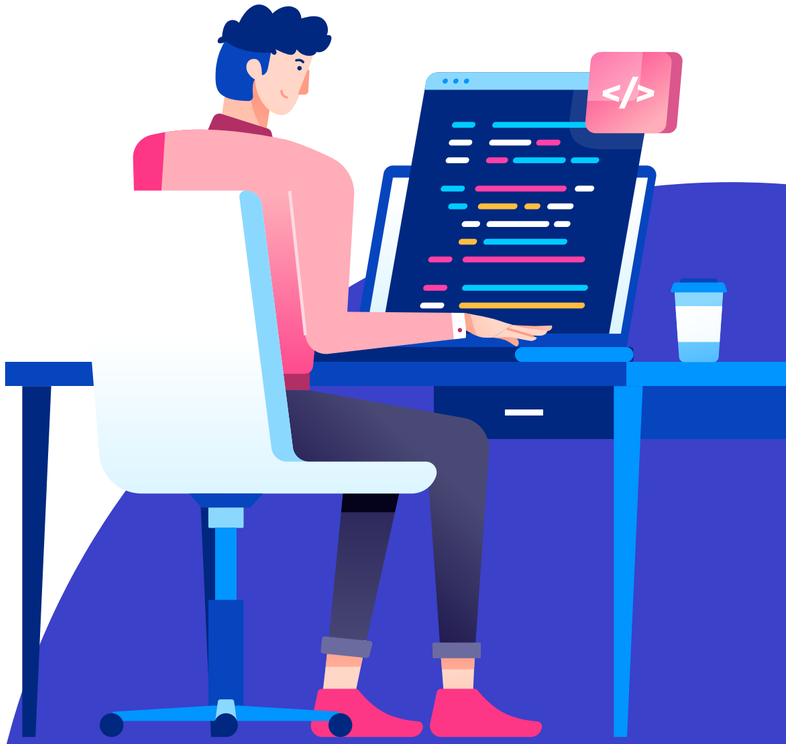
2. Нове значення змінної витісняє старе. Важливо уявляти, чому дорівнює значення змінної в кожен момент часу.

3. Як назву змінних заборонено використовувати ключові (зарезервовані) слова. До ключових слів у мові Python належать:

False;
True;
None;
and;
with;
as;
assert;
break;
class;
continue;
def;
del;
elif;
else;
except;
finally;
try;
for;
from;
global;
if;
import;
in;
is;
lambda;
nonlocal;
not;
or;
pass;
raise;
return;
while;
field.

ЗАНЯТТЯ 3

ЦІЛОЧИСЛОВА АРИФМЕТИКА



ДОДАТКОВІ ОПЕРАЦІЇ

У Python над числами можна здійснювати **3 додаткові операції:**

Операція	**Опис**	**Пріоритет**
**	піднесення до степеня	0
//	цілочислове ділення	1
%	знаходження остачі від ділення	1

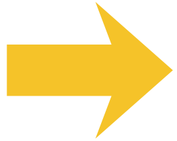


Для зміни пріоритету потрібні дужки

ПІДНЕСЕННЯ ДО СТЕПЕНЯ

Операція піднесення до степеня **a**n** підносить число **a** до степеня **n**:

```
print(2 ** 0)
print(2 ** 1)
print(2 ** 2)
print(2 ** 3)
print(2 ** (-1))
```



```
1
2
4
8
0.5
```

На відміну від багатьох мов програмування, в Python для піднесення до степеня використовується символ **, а не ^

ЦІЛОЧИСЛОВЕ ДІЛЕННЯ

Операція цілочислового ділення **//** відкидає десяткову частину результата

```
print(10 // 3)
print(10 // 4)
print(10 // 5)
print(10 // 6)
print(10 // 12)
print(-10 // 12)
```

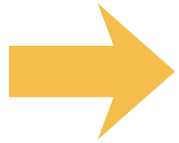
```
3
2
2
1
0
-1
```

При цілочисловому діленні округлення здійснюється в менший бік

ЗНАХОДЖЕННЯ ОСТАЧІ ВІД ДІЛЕННЯ

Операція знаходження остачі **%** повертає остачу від ділення двох цілих чисел:

```
print(10 % 3)
print(10 % 4)
print(10 % 5)
print(10 % 6)
print(10 % 12)
print(3 % 7)
```



```
1
2
0
4
10
3
```

Число ділиться на n націло, якщо **остача** від ділення на n **дорівнює 0**



Остача, при діленні на число **n,** може набувати значень: **0, 1, 2,..., n-1**

РОЗБЕРЕМО ...

- 1 ** 10 = 1
- 2 ** 3 = 8
- 3 ** 2 = 9
- 5 ** 4 = 625
- 9 ** (1/2) = 3
- 2 ** (-2) = 0.25

РОЗБЕРЕМО ...

- 24 // 5 = 4
- 34 // 3 = 11
- 100 // 12 = 8
- 12 // 90 = 0
- -17 // 8 = -3

РОЗБЕРЕМО ...

- 24 % 5 = 4
- 34 % 3 = 1
- 10 % 15 = 10
- 120 % 90 = 30
- 9 % 90 = 9
- -17 % 8 = 7

ОБРОБКА ЦИФР ЧИСЛА

Обробка цифр числа – робота з окремими цифрами одного числа (наприклад: сума цифр, кількість цифр, найбільша цифра, перевірка умов).

Це потрібно для:

- аналізу даних;
- перевірки правильності введених чисел (PIN-коди, ID, контрольні суми) тощо.

ЦИФРИ ДВОЗНАЧНОГО ЧИСЛА

За допомогою операцій **//** та **%** можна знаходити окремі цифри числа

```
num = 17
a = num % 10
b = num // 10
print(a)
print(b)
```

```
7
1
```

```
num = 754
a = num % 10
b = (num % 100) // 10
c = num // 100
print(a)
print(b)
print(c)
```

```
4
5
7
```

Отримати останню цифру числа **n: n % 10**
Відкинути останню цифру від числа **n: n // 10**

АЛГОРИТМ ЗНАХОДЖЕННЯ ЦИФР

для **n-значного** числа **num**:

- остання цифра: **(num % 10**1) // 10**0;**
- передостання цифра: **(num % 10**2) // 10**1;**
- третя з кінця цифра цифра: **(num % 10**3) // 10**2;**
- .....
- друга цифра: **(num % 10**(n-1)) // 10**(n-2);**
- перша цифра: **(num % 10**n) // 10**(n-1)**

ЗАДАЧА 1. ТРИЗНАЧНЕ ЧИСЛО

Написати програму, в якій розраховується сума та добуток цифр додатного тризначного числа.

Вхідні дані	Вихідні дані
342	Сума цифр = 9 Добуток цифр = 24

```
num = int(input())
a = num % 10
b = (num % 100) // 10
c = num // 100
print(a + b + c)
print(a * b * c)
```

```
342
9
24
```

ЗАНЯТТЯ 4

БАЗОВІ АЛГОРИТМІЧНІ СТРУКТУРИ: СТРУКТУРА СЛІДУВАННЯ



1. a = n % 10

2. a = n // 10

3. a = 5 % 10

4. a = 10 // 5

Обери правильний варіант

записує у змінну a результат ділення n на 10	записує у змінну a останню цифру числа n	записує у змінну a першу цифру числа n
записує у змінну a першу цифру числа n	записує у змінну a число n без першої цифри	записує у змінну a число n без останньої цифри
0	5	10
10	2	0

БАЗОВІ АЛГОРИТМІЧНІ СТРУКТУРИ

- слідування
- розгалуження
- повторення

Програма – *це набір інструкцій, написаних мовою програмування, які визначають* ***порядок*** *виконання операцій для розв’язання певного завдання*

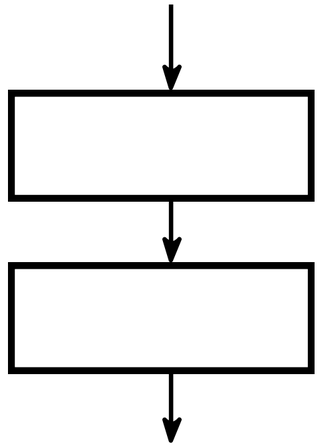
Слідування — це форма організації команд (вказівок) в алгоритмі (програмі), під час виконання яких дії виконуються ***послідовно***, одна за одною, без пропусків або повторень.



ЛІНІЙНІ АЛГОРИТМИ

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.

Основою лінійних алгоритмів є базова алгоритмічна структура **слідування.**



Структура лінійного алгоритму

ПРОГРАМА

Програма - це набір інструкцій мовою програмування, які описують певну послідовність дій

- Інструкції завжди виконуються послідовно одна за одною, згори донизу
- У свою чергу кожна інструкція створює або обробляє об'єкти

РОЗБЕРЕМО ...

```
print (5 + 6)
```

Відбувається додавання двох об'єктів: ціле число 5 додається до цілого числ 6. У результаті створюється новий об'єкт (число 11), який і буде виведено на екран.

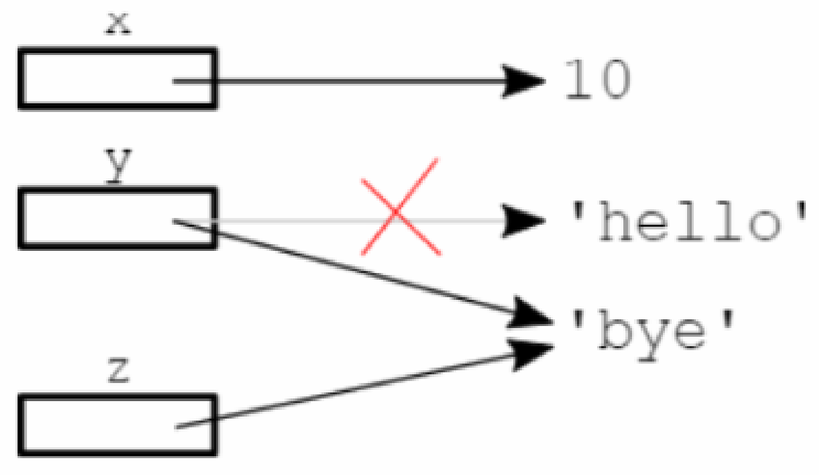
Поняття «**об'єкт**» є ключовим у мові Python, оскільки будь-яке значення в програмі є об'єктом

ТИПИ ОБ’ЄКТІВ В PYTHON

Типи об’єктів	Назва в Python	Приклад
Ціле число	int (від “integer”)	43; -56; 0; 2548
Число з плаваючою крапкою (дійсні числа)	float	1.5; -43.0; 32.5
Рядок	str (від “string”)	‘hello’, “Hello”
Логічний тип	bool (від “boolean”)	True, False

ПРИКЛАДИ ОГОЛОШЕННЯ ЗМІННИХ

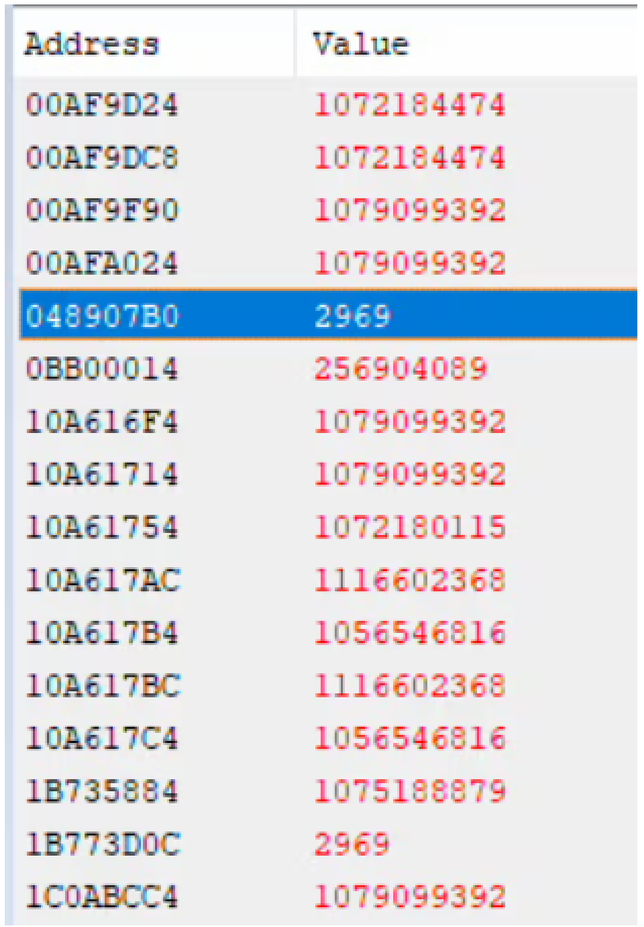
```
x = 10
y = 'hello'
y = 'bye'
z = y
```



Тип даних у мові Python визначається транслятором автоматично **не за іменем змінної, а за значенням**, яке зберігається у цій змінній.

ІМЕНОВАНА ОБЛАСТЬ ПАМ'ЯТІ

- **Змінна** - іменована область пам'яті, призначена для зберігання значення.
- Змінна створюється в момент першого присвоювання їй значення.
- Змінна в Python не зберігає саме значення, вона зберігає **посилання (адресу) на об'єкт.**



Address	Value
00AF9D24	1072184474
00AF9DC8	1072184474
00AF9F90	1079099392
00AFA024	1079099392
048907B0	2969
0BB00014	256904089
10A616F4	1079099392
10A61714	1079099392
10A61754	1072180115
10A617AC	1116602368
10A617B4	1056546816
10A617BC	1116602368
10A617C4	1056546816
1B735884	1075188879
1B773D0C	2969
1C0ABCC4	1079099392

СКЛАДОВІ ЗМІННОЇ

1. Ім'я змінної (ідентифікатор)
2. Значення змінної
3. Тип змінної (тип об'єкта, який зберігається у змінній)

ПРИКЛАД ...

```
1. print(type(10))      # <class 'int'>
2. print(type(10.5))    # <class 'float'>
3. print(type(0))       # <class 'int'>
4. print(type(0.0))     # <class 'float'>
5. print(type(-34))     # <class 'int'>
6. print(type(-43.7)) # <class 'float'>
```

Якщо за записом ви не можете визначити тип об'єкта, користуйтеся функцією **type**

ТИПИ ЗМІННИХ В МАТЕМАТИЧНИХ ДІЯХ

Операнд	Дія	Операнд	Результат
Ціле	+	Ціле	Ціле
Дійсне	+	Ціле	Дійсне
Ціле	-	Ціле	Ціле
Ціле	*	Ціле	Ціле
Ціле	*	Дійсне	Дійсне
Будь-яке	/	Будь-яке	Дійсне
Ціле	**	Ціле	Ціле
Ціле	**	Дійсне	Дійсне

ВБУДОВАНІ ФУНКЦІЇ

Функція	Результат
abs (value)	abs (-7) -> 7, abs (7) -> 7
min (x1, x2, … xn)	min (-43, 90, 832, -78) -> -78
max (x1, x2, … xn)	max (-43, 90, 832, -78) -> 832
pow (x, y)	pow (2, 5) -> 32
round()	round(3.5) -> 4 round(2.5) -> 2

round()
парні числа округляються в менший бік, непарні - в більший

МОДУЛЬ MATH

... надає функціонал для обчислень з дійсними числами

Способи підключення

1.
```
import math
# програмний код ...
```

```
import math
num1 = math.sqrt(2)      # обчислення квадратного кореня з двох
num2 = math.ceil(3.8)    # округлення числа вгору
num3 = math.floor(3.8)   # округлення числа вниз
```

2.
```
from math import *
```

3.
```
from math import sqrt, ceil
```

```
from math import *

num1 = sqrt(2)
num2 = ceil(3.8)
num3 = floor(3.8)
```

ДЕЯКІ ФУНКЦІЇ МОДУЛЯ MATH

Функція	Опис
ceil(x)	Повертає найменше ціле число, більше або рівне x
fabs(x)	Повертає абсолютне значення x
factorial(x)	Повертає факторіал x
floor(x)	Повертає найбільше ціле число, менше або рівне x
fmod(x, y)	Повертає остачу від ділення x на y
modf(x)	Повертає дробову та цілу частини x
trunc(x)	Повертає “зрізане” цілочисельне значення x (відкидає дробову частину числа)

ДЕЯКІ ФУНКЦІЇ МОДУЛЯ MATH

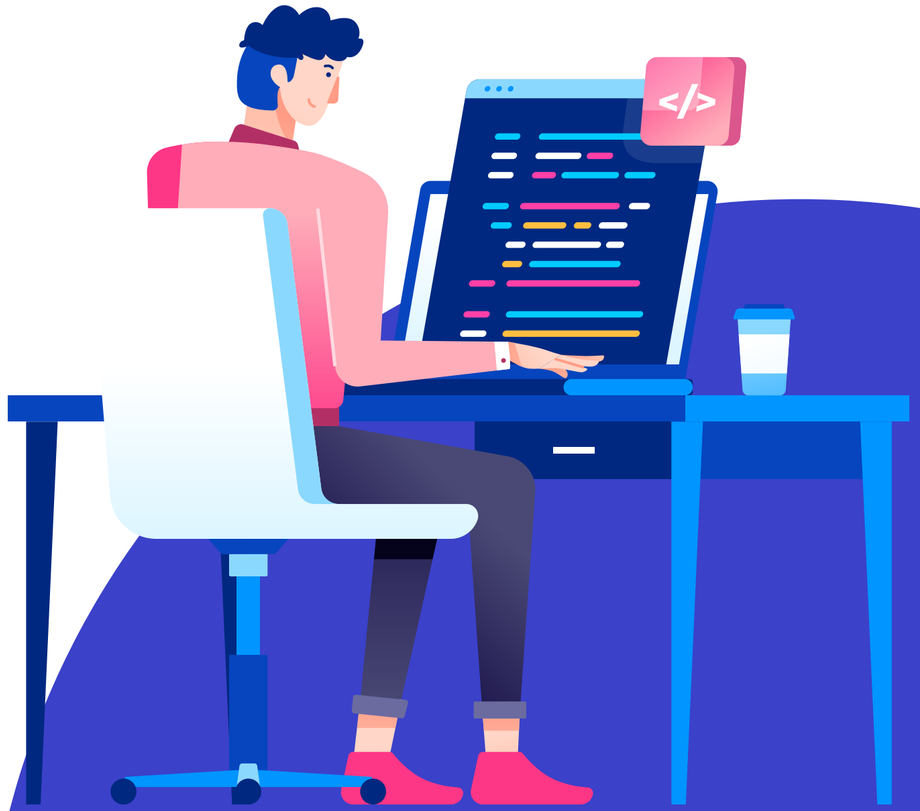
degrees(x)	Перетворення кута x з радіанів у градуси
radians(x)	Перетворення кута x з градусів у радіани
cos(x)	Повертає косинус x
sin(x)	Повертає синус x
tan(x)	Повертає тангенс x
hypot(x, y)	Обчислює гіпотенузу трикутника з катетами x та y (math.sqrt(x * x + y * y))
pi	Математична константа *Пі*, що дорівнює відношенню довжини кола до його діаметра (3,14159...)
e	Математична константа e (2,71828...)

ДЕЯКІ ФУНКЦІЇ МОДУЛЯ MATH

exp(x)	Повертає e**x
log(x[, b])	Повертає логарифм x за основою b (за замовчуванням e)
log2(x)	Повертає логарифм x за основою 2
log10(x)	Повертає логарифм x за основою 10
pow(x, y)	Повертає x, піднесений до степеня y
sqrt(x)	Повертає квадратний корінь з x
acos(x)	Повертає арккосинус x
asin(x)	Повертає арксинус x
atan(x)	Повертає арктангенс x
atan2(y, x)	Повертає значення atan(y/x) (в радіанах)

ЗАНЯТТЯ 5

ЛОГІЧНІ ОПЕРАЦІЇ



1. a = abs(-3.8)

2. a = sqrt(16)

3. a = ceil(-4.3)

4. a = pow(2,4)

Обери правильний варіант

-3.8	-3	3.8	4
4	8	256	
-5	-4	-3	4
6	8	16	

ЛОГІЧНИЙ ТИП ДАНИХ (BOOLEAN)

… примітивний тип даних, який може набувати лише два можливі значення: **істинність (true)** і **хибність (false)**.

У Python логічний тип даних реалізований під назвою **bool.**

Вбудований тип даних bool в мові Python може набувати лише одне з **двох** значень:

✔️ True (Істина);
✔️ False (Хибність)

1. У Python-програмі обов’язково потрібно використовувати ці значення **з великої літери**.
2. Результатом будь-якого порівняння або умови буде значення логічного типу: **True** або **False**.

ПЕРЕТВОРЕННЯ В ЛОГІЧНИЙ ТИП ДАНИХ

int/float → bool

```
print(bool(0))  # False
print(bool(1))  # True
print(bool(1.0))  # True
print(bool(-42))  # True
print(bool(345.3))  # True
print(bool(-123.35))  # True
```

str → bool

```
print(bool(""))        # False
print(bool("True"))    # True
print(bool("False"))   # True
print(bool("0"))       # True
print(bool(" "))       # True
```

bool → int

```
print(int(True))   # 1
print(int(False))  # 0
```

ОПЕРАТОРИ ПОРІВНЯННЯ

Для перевірки умови використовують спеціальний символ:
подвійне дорівнює (==):

```
print ('Яку мову програмування ми вивчаємо?')
answer = input()
if answer == 'Python':
    print('Так, ми вчимо Python')
    print('Python - чудова мова!')
```



== це **умовний оператор порівняння** (перевірка на рівність)

ОПЕРАТОРИ = VS ==

Не плутати:

оператор **присвоювання** (=) та **умовний оператор** (==):

```
num = 2025
s = 'I love Python'
```

оператор присвоювання (=) надає змінним значеннння

```
if answer == 'Python':
if name == 'Gvido':
if temperature == 25:
```

умовний оператор (==) перевіряє на рівність два значення



Плутанина з операторами == та = є однією з найрозповсюдженіших та “неприємних” помилок в програмуванні

ОПЕРАТОРИ ПОРІВНЯННЯ

У мові Python існує **6** операторів порівняння:

Оператор	Опис	Приклад
>	більше	if x > 7:
<	менше	if x < 7:
>=	більше або дорівнює	if x >= 7:
<=	менше або дорівнює	if x <= 7:
==	дорівнює	if x == 7:
!=	не дорівнює	if x != 7:

ЛАНЦЮГИ ПОРІВНЯНЬ

a **==** ***b*** **==** ***c***
1 <= ***x*** **<= 10**

```
age = int(input())
if 3 <= age <= 6:
    print('Дитина')
```

Операція рівності є **транзитивною:**
якщо ***a*** **==** ***b*** і ***b*** **==** ***c***, то ***a*** **==** ***c***.

ЛОГІЧНІ ОПЕРАЦІЇ

Складена умова складається з декількох умов.
В мові Python є три логічні операції, які дозволяють створювати складені умови:

1. **and**: логічне множення, “і”
2. **or**: логічне додавання, “або”
3. **not**: логічне заперечення, “ні”

ТАБЛИЦЯ ІСТИНОСТІ ДЛЯ ОПЕРАЦІЇ AND

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	**True**

Операція **and** повертає **True**,
коли обидва вирази істинні

ТАБЛИЦЯ ІСТИНОСТІ ДЛЯ ОПЕРАЦІЇ OR

a	b	a or b
False	False	**False**
False	True	True
True	False	True
True	True	True

Операція **or** повертає **False**, коли обидва вирази хибні

ТАБЛИЦЯ ІСТИНОСТІ ДЛЯ ОПЕРАЦІЇ **NOT**

a	not a
False	True
True	False



Операція **not** має найвищий пріоритет

ТАБЛИЦЯ ІСТИНОСТІ ДЛЯ ОПЕРАЦІЇ XOR

a	b	a xor b
False	False	**False**
False	True	True
True	False	True
True	True	**False**

Операція **xor** повертає **False**,
коли обидва вирази **однакові**

ПРІОРИТЕТ ЛОГІЧНИХ ОПЕРАЦІЙ

Операція	Пріоритет
not	0
and	1
or, xor	2



Для явного визначення *порядку* виконання логічних операцій використовують **дужки**

ЗВЕРНІМО УВАГУ НА ...

1. Плутанину логічних операторів AND і OR

```
if x > 1 and x < 100:
if x > 1 or x < 100:
```

2. Синтаксис запису команд

```
if age >= 7 and <= 9:
```

```
if age >= 7 and age <= 9:
```

3. Ланцюги порівнянь

```
if age >= 7 and age <= 9:
```

=

```
if 7 <= age <= 9:
```

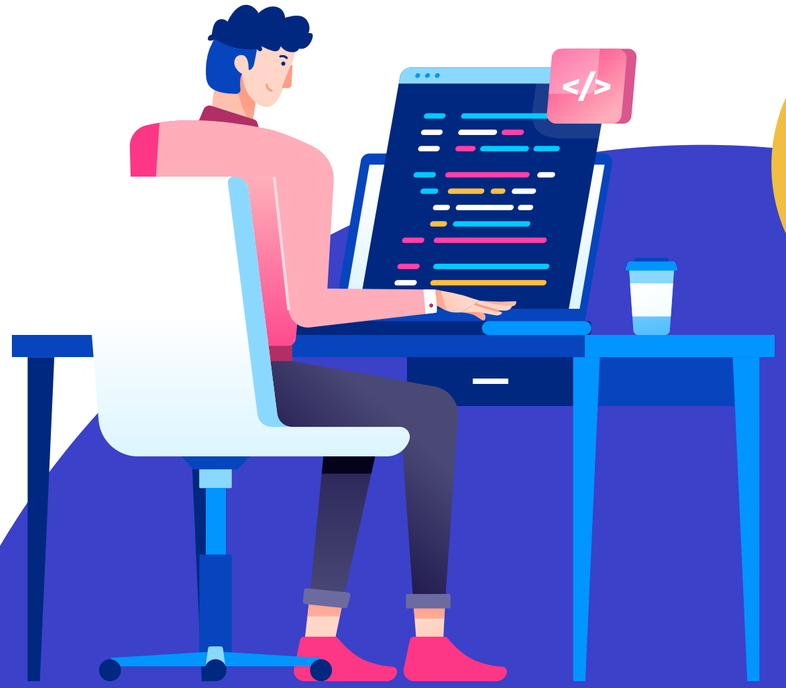
4. Скорочену схему порівнянь

```
5 > 100 and 10 > 0
```

```
10 > 0 or 5 > 100
```

ЗАНЯТТЯ 6

СТРУКТУРА РОЗГАЛУЖЕННЯ



1. True and False

2. False or False

3. a = not 12

4. 5 and -5

Обери правильний варіант

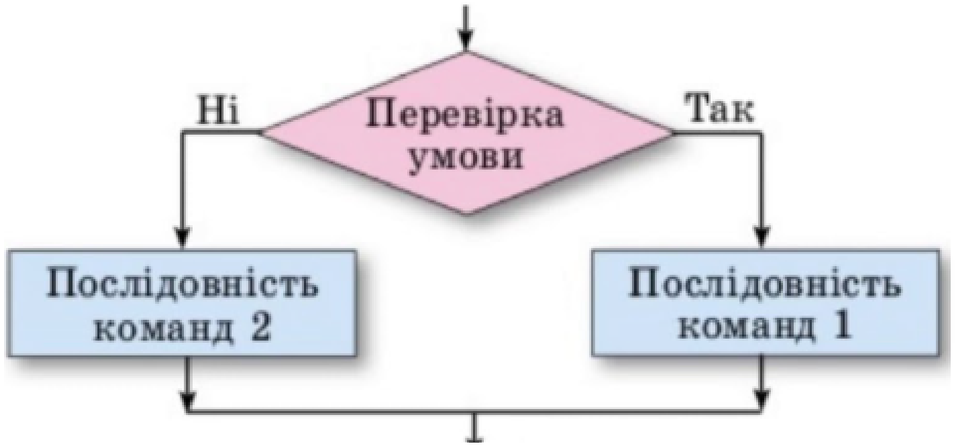
True	or	False	not
True	and	False	not
11	True	0	False
True	False	5	-5

ОПЕРАТОР РОЗГАЛУЖЕННЯ

… конструкція **if-elif-else**, яка дозволяє програмі вибирати, який блок коду виконувати, залежно від істинності певних умов, використовуючи ключові слова **if (якщо)**, **elif (інакше-якщо)** та **else (інакше)**.

УМОВНИЙ ОПЕРАТОР

Для прийняття рішення програма перевіряє істинність або хибність певної умови



Перевірка умови та прийняття рішень за результататми цієї перевірки наз. **розгалуженням** (**branching**). Програма в такий спосіб обирає, за якою з можливих гілок їй рухатися далі

УМОВНИЙ ОПЕРАТОР

Програма зчитує текст, перевіряє, чи введений текст дорівнює рядку 'Python'
якщо - “так”, то виводить користувачеві текст

```
print ('Яку мову програмування ми вивчаємо?')
answer = input()
if answer == 'Python':
    print('Так, ми вчимо Python')
    print('Python - чудова мова!')
```

Двокрапка (:) *наприкінці рядка повідомляє, що далі знаходиться блок команд.*
У блок входять усі рядки, розташовані з відступом.

Результат роботи програми:

```
Яку мову програмування ми вивчаємо?
Python
Так, ми вчимо Python
Python - чудова мова!
```

УМОВНИЙ ОПЕРАТОР

```
print ('Яку мову програмування ми вивчаємо?')
answer = input()
if answer == 'Python':
    print('Так, ми вчимо Python')
    print('Python - чудова мова!')
```



відступ **блок команд**

В Python **відступ** - це важлива частина коду.

Відступ - невеликий зсув рядка кода вправо.

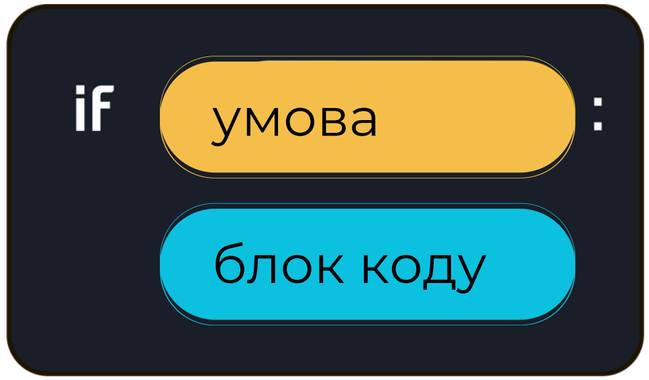
Відступ вказує, де починається і де завершується **блок коду**

PEP 8

Для відступів блоків коду застосовують 4 пробіли

УМОВНИЙ ОПЕРАТОР

Скорочена форма:



Повна форма:



УМОВНИЙ ОПЕРАТОР (ПОВНА ФОРМА)

```
print ('Яку мову програмування ми вивчаємо?')
answer = input()
if answer == 'Python':
    print('Так, ми вчимо Python')
    print('Python - чудова мова!')
else:
    print('То, напевно, не з нами')
```

Результат роботи програми:

```
Яку мову програмування ми вивчаємо?
Pascal
То, напевно, не з нами
```

ЗАДАЧА

Напишіть програму, яка для заданого цілого числа x визначає, чи належить це число вказаному проміжку.



Формат вхідних даних
На вхід до програми подається ціле число x.

Формат вихідних даних
Програма має вивести текст “Так, належить” або “Ні, не належить” (без лапок).

Результат роботи програми:

```
x = int(input())
if -1 < x <17:
    print('Так, належить')
else:
    print('Ні, не належить')
```

```
12
Так, належить
```

```
-10
Ні, не належить
```

ТЕРНАРНИЙ ОПЕРАТОР

Тернарний оператор – це компактний спосіб запису умовних конструкцій у Python.

true_value **if** condition **else** false_value

condition *- умова*
rue_value *- значення, якщо задана умова (condition) є істинною*
false_value *- значення, якщо задана умова (condition) є хибною*

```
print(5) if 3%2==1 else print(10)
```

```
if 5 % 2 == 1:
    ans = "Так"
else:
    ans = "Ні"
```



```
ans = "Так" if 5 % 2 == 1 else "Ні"
```

ПОВНА ФОРМА VS ТЕРНАРНИЙ ОПЕРАТОР

Визначити ціну квитка залежно від віку відвідувача. Для повнолітнього відвідувача ціна становитиме 20 гривень, для решти – 8.

```
age = int(input('Введіть вік: '))
if age >= 18:
    price_ticket = 20
else:
    price_ticket = 8
print(f"Ціна білета {price_ticket} грн")
```

Зверніть увагу на форматоване виведенння

```
age = int(input('Введіть вік: '))
price_ticket = 20 if age >= 18 else 8
print(f"Ціна білета {price_ticket} грн")
```

ВКЛАДЕНИЙ УМОВНИЙ ОПЕРАТОР

Всередині блоку умовного оператора можна використовувати **будь-які** інструкції мови Python, зокрема інший умовний оператор.

Отримаємо **вкладене розгалуження** - всередині гілки виникає ще одна гілка.

```
if умова_1:
    блок кода
else:
  if умова_2:
     блок кода
  else:
     if умова_3:
        блок кода
...
```

```
if умова1:
    if умова2:
       блок команд 1
    else:
       блок команд 2
else:
    блок команд 3
...
```



Вкладені блоки мають великі розміри відступу (8, 12, ... пробілів)

ВКЛАДЕНИЙ УМОВНИЙ ОПЕРАТОР

```
a = int(input())
if a % 2 == 0:
    if a % 5 == 0:
        print(a, 'ділиться на 2 і 5')
    else:
        print(a, 'ділиться на 2, але не на 5')
else:
    print(a, 'не ділиться на 2')
```

```
20
20 ділиться на 2 і 5
```

```
12
12 ділиться на 2, але не на 5
```

```
15
15 не ділиться на 2
```

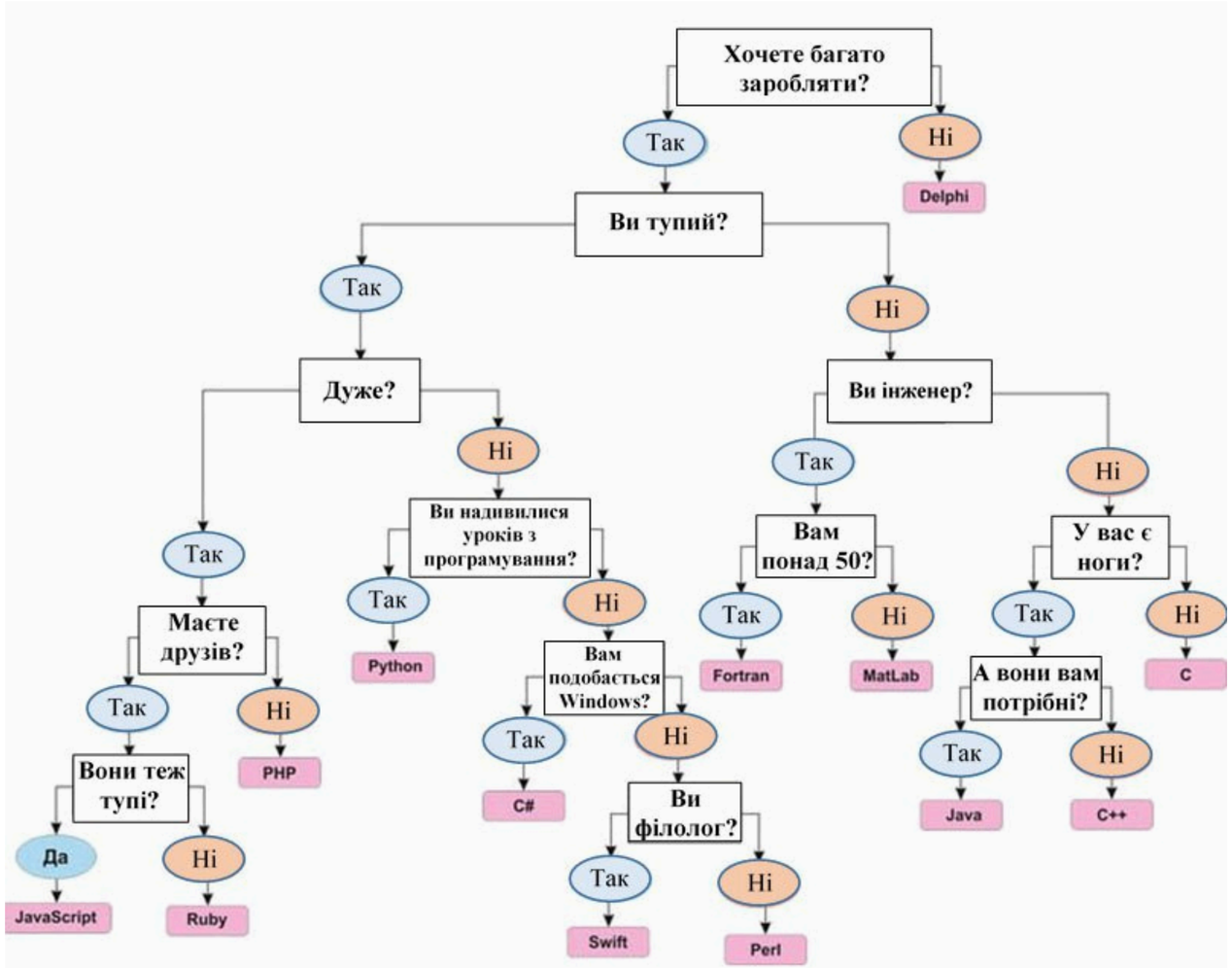
ОБМЕЖЕННЯ ПО ВКЛАДЕНОСТІ УМОВ

Жодних обмежень за рівнями вкладності умов не має.
Можна вкладати скільки завгодно умов одна в одну, при цьому не **не забуваючи** створювати **нові рівні відступу.**

АЛЕ є правила **дзена**, за одним з яких:

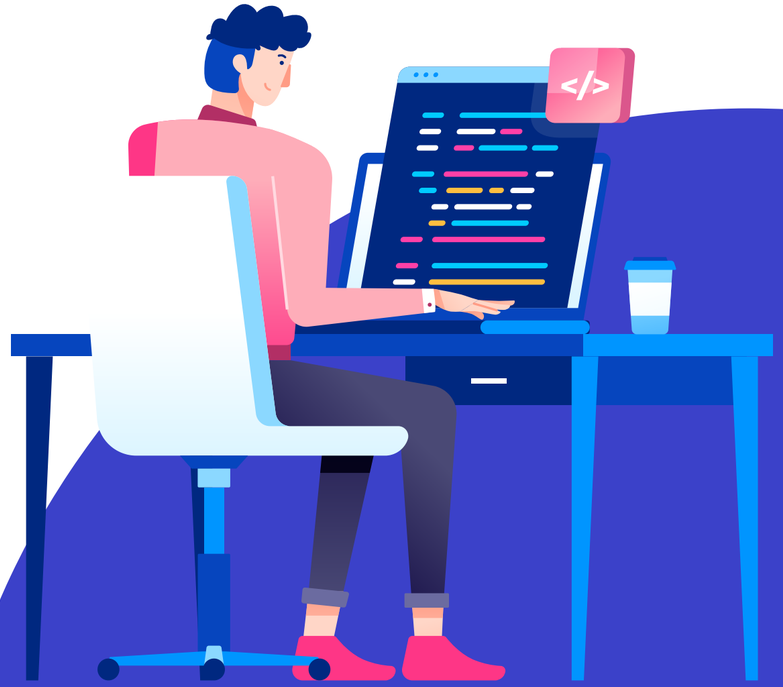
> Пласке краще, ніж вкладене

Краще уникати вкладених умов, якщо є така можливість



ЗАНЯТТЯ 7

СТРУКТУРА ВИБОРУ



Що буде результатом роботи програми, якщо користувач введе 50?

```
a = int(input())
if a > 1:
    print(">1")
if a > 10:
    print(">10")
if a > 100:
    print(">100")
print('Кінець')
```

А	Кінець
Б	>1 >10 Кінець
В	Помилка IndentationError
Г	>1 Кінець
Д	>1 >10 >100 Кінець

Що буде результатом роботи програми?

```
a = int(input())
if a > 1:
    print(">1")
if a > 10:
    print(">10")
if a > 100:
    print(">100")
print('Кінець')
```

А	Кінець
Б	>1 >10 Кінець
В	Помилка IndentationError
Г	>1 Кінець
Д	>1 >10 >100 Кінець

Що буде результатом роботи програми?

```
if True:
    print("True")
if bool(-90):
    print(-90)
if "0":
    print(0)
print('Завершено')
```

А	True 0 Завершено
Б	True -90 0 Завершено
В	Завершено
Г	True -90 Завершено
Д	True Завершено

КАСКАДНИЙ УМОВНИЙ ОПЕРАТОР

У мові Python використовується для перевірки декількох умов

```
if умова_1:
    блок коду_1
elif умова_2:
    блок коду_2
elif умова_3:
    блок коду_3
...
else:
  блок коду
```

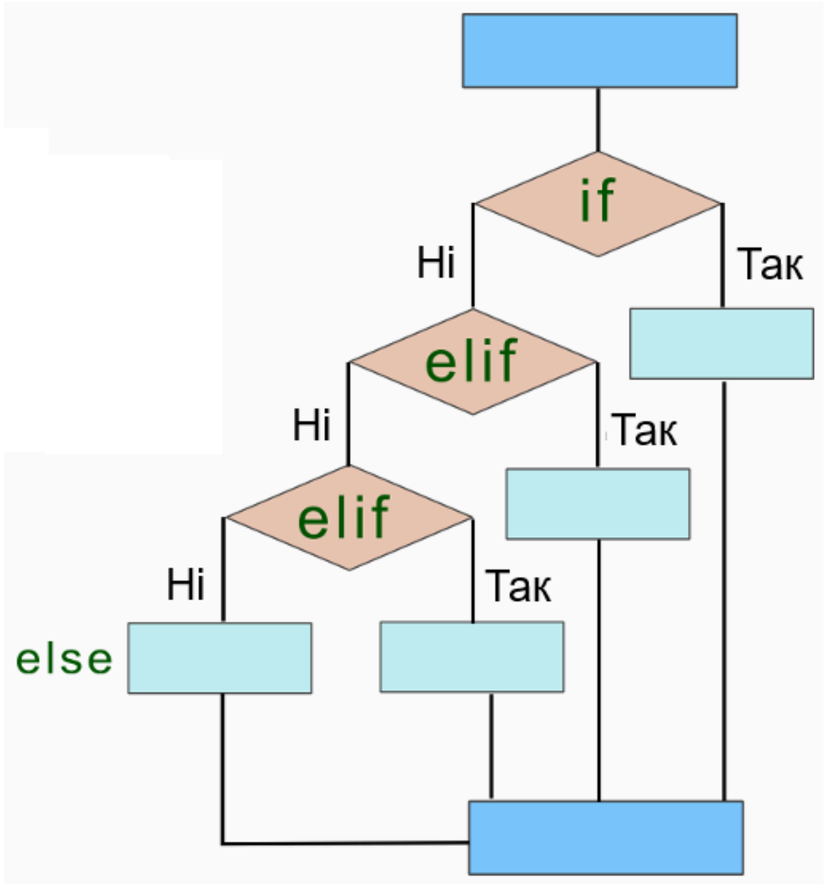
1. Спочатку перевіряється умова _1:
   - Якщо вона **істина**, то виконується **блок коду_1**
   - Якщо **хибна** - переходимо до **умова _2**
2. Перевіряємо умова _2:
   - Якщо вона **істина**, то виконується **блок коду_2**
   - Якщо **хибна** - переходимо до **умова _3**

...

3. Процес продовжується доки, поки не буде знайдена істинна умова, або доки не закінчаться вирази **elif**
4. Якщо жодна умова не є істинною, то виконується блок коду, записаний після **else**

КАСКАДНИЙ УМОВНИЙ ОПЕРАТОР



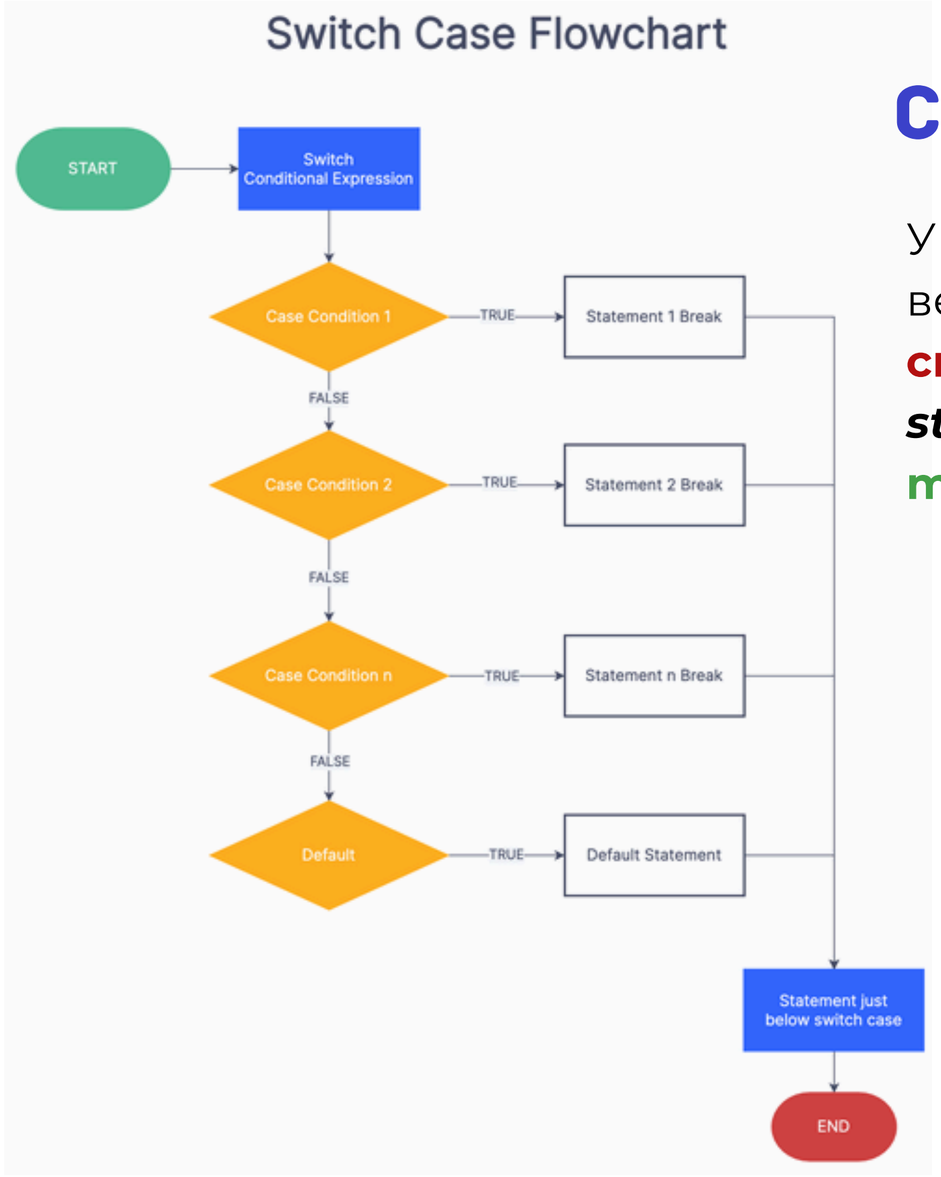


ЗАУВАЖИМО!

1. Каскадний оператор **if - elif - else** може бути запрограмований вкладеними операторами, зокрема **if - else**

2. Каскадний оператор **if - elif - else** зазвичай простіше, ніж довга серія вкладених операторів **if - else**

3. Заключний блок **else** в каскадному операторі **if - elif - else** є **необов'язковим!**



ОПЕРАТОР СПІВСТАВЛЕННЯ З ШАБЛОНАМИ

У мові Python аналог ***switch-case*** з'явився тільки у версії Python 3.10. Він називається **оператором співставлення з шаблонами** (***pattern matching statement***), замість *switch-case* використовується **match-case**.

```
match value:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
     <action_wildcard>
```

value - *значення, яке потрібно перевірити на співпадіння;*

pattern_n - *шаблон, з яким порівнюється;*

action_n - *дія, яка виконається, якщо співпадіння відбудеться*

case _ - *виконається, якщо не знайдеться жодного співпадіння (не обов'язковий аналог default)*

ОПЕРАТОР СПІВСТАВЛЕННЯ З ШАБЛОНАМИ

```
digit = int(input("Введіть цифру: "))
match digit:
    case 0 | 3 | 6 | 9:
        print("Без остачі діляться на 3")
    case 1 | 4 | 7:
        print("При діленні на 3 дають остачу 1")
    case 2 | 5 | 8:
        print("При діленні на 3 дають остачу 2")
    case  _:
        print(f"{digit} не є цифрою")
```



Оператор ***match-case*** порівнює значення з шаблоном тільки за рівністю (аналог **==**). Більш складні умови, такі як перевірка діапазону (**>**, **<**) або комбіновані умови, в **match-case** недоступні. Для таких випадків використовуються **if-elif-else.**

МОРЖОВИЙ ОПЕРАТОР



За допомогою **моржового оператора :=** можна одночасно надати значення змінній та використати це значення. Це скорочує кількість рядків коду та робить його більш лаконічним.

Як це працює:

1️⃣ Оператор **:=** виконує присвоювання значення змінній **see_walrus** всередині виклику **print();**

2️⃣ Відразу після цього значення змінної підставляється у функцію **print();**

3️⃣ Змінна **see_walrus** зберігає присвоєне значення та доступна для подальшого використання.

```
see_walrus = 'Look at my walrus'
print(see_walrus)
```

⟺

```
print(see_walrus := 'Look at my walrus')
```

ЗАНЯТТЯ 8

СТРУКТУРА ПОВТОРЕННЯ: ЦИКЛ FOR



ЯК ВИ РОЗУМІЄТЕ ВИСЛІВ “КОДИТИ РОЗУМНО”?



Розумне програмування — як інженерне мислення: оптимізуй, автоматизуй, адаптуй

Що буде результатом роботи програми?

```
score = 74
if score < 60:
    print("F")
elif score >= 60:
    print("D")
elif score >= 70:
    print("C")
elif score >= 80:
    print("B")
elif score >= 90:
    print("A")
```

АДАПТУЙ!

А	F
Б	D
В	C
Г	B
Д	A

Що буде результатом роботи програми?

```
score = 74
if score < 60:
    print("F")
elif score >= 60:
    print("D")
elif score >= 70:
    print("C")
elif score >= 80:
    print("B")
elif score >= 90:
    print("A")
```

АДАПТУЙ!

```
score = 74
if score >= 90:
    print("A")
elif score >= 80:
    print("B")
elif score >= 70:
    print("C")
elif score >= 60:
    print("D")
else:
    print("F")
```

```
score = 74
if score >= 90:
    print("A")
else:
    if score >= 80:
        print("B")
    else:
        if score >= 70:
            print("C")
        else:
            if score >= 60:
                print("D")
            else:
                print("F")
```

```
score = 74
match score:
    case s if s >= 90:
        print("A")
    case s if s >= 80:
        print("B")
    case s if s >= 70:
        print("C")
    case s if s >= 60:
        print("D")
    case _:
        print("F")
```

Що буде результатом роботи програми?

```
value = 3.5
match value:
    case int():
        print("Це число")
    case str():
        print("Це рядок")
    case _:
        print("Невідомий тип")
```

ОПТИМІЗУЙ!

А	Це число
Б	Це рядок
В	Невідомий тип
Г	Помилка виконання
Д	Дійсне число

Що буде результатом роботи програми?

```
value = 3.5
match value:
    case int():
        print("Це число")
    case str():
        print("Це рядок")
    case _:
        print("Невідомий тип")
```

ОПТИМІЗУЙ!

```
value = 3.5
if value == int(value):
    print("Це число")
elif value == str(value):
    print("Це рядок")
else:
    print("Невідомий тип")
```

Що буде результатом роботи програми?

```
number = 8
match number % 2:
    case 0:
        print("Парне число")
    case 1:
        print("Непарне число")
    case _:
        print("Це число")
```

ОПТИМІЗУЙ!

А	Парне
Б	Непарне число
В	Не шаблон
Г	Помилка виконання

Що буде результатом роботи програми?

```
number = 8
match number % 2:
    case 0:
        print("Парне число")
    case 1:
        print("Непарне число")
    case _:
        print("Це число")
```

ОПТИМІЗУЙ!

```
number = 8
if number % 2 == 0:
    print("Парне число")
else:
    print("Непарне число")
```

```
number = 8
ans = "Парне" if number%2 == 0 else "Непарне"
print(ans)
```

```
number = 8
print(ans := "Парне" if number%2 == 0 else "Непарне")
```

АВТОМАТИЗУЙ!

e-olymp

https://eolymp.com

Система ***E-olymp*** — це український інтернет-портал та інформаційно-телекомунікаційна система для дистанційної підготовки учнів, студентів та вчителів до олімпіад з програмування, яка надає можливість розв'язувати задачі, брати участь у змаганнях та тестувати свої розв'язки в різних мовах програмування, підтримуючи самоосвіту та підвищення рівня IT-компетентностей в Україні та світі.

ОПЕРАТОРИ ЦИКЛУ

Оператори циклу в програмуванні дозволяють виконувати блок коду доки виконується задана умова або вказану кількість разів.

ЦИКЛИ

Перевага комп’ютерів перед людьми - здатність повторювати однакові дії багатократно.

У мові Python є **два** основні **різновиди циклів**:

- цикли, які повторюються певну кількість разів (**for, цикл з лічильником, counting loops**) - застососується для відомої кількості повторень / ітерацій;
- цикли, які повторюються доки виконується задана умова (**while, умовні цикли, conditional loops**) - застососується, коли кількість повторень / ітерацій зазделегідь невідома або залежить від умови.

ОПЕРАТОР FOR

Цикл **for** – це керуюча конструкція в мові програмування Python, яка забезпечує послідовне виконання блоку команд для кожного елемента заданої послідовності

ЦИКЛ FOR

Синтаксис цикла **for** в мові Python:

```
for назва змінної in range(кількість повторень):
    блок кода
```



Двокрапка **(:)** в кінці рядка повідомляє Python, що далі знаходиться блок команд, який називається **тілом цикла.**

ПРИКЛАД

Порахувати квадрат введеного числап’ять разів.

```
for i in range(5):
    num = int(input())
    print(f"Квадрат введеного числа {num} =", num * num)
```

Цикл повторюється 5 разів

```
43
Квадрат введеного числа 43 = 1849
6
Квадрат введеного числа 6 = 36
3
Квадрат введеного числа 3 = 9
9
Квадрат введеного числа 9 = 81
2
Квадрат введеного числа 2 = 4
```

ЦИКЛ FOR VS УМОВНИЙ ОПЕРАТОР IF

часто використо-вуються разом

Структура цикла **for**:

```
for назва змінної in range(кількість повторень):
    блок кода
```

for застосовують тоді, коли потрібно повторити одну й ту саму дію кілька разів

Структура умовного оператора **if:**

```
if умова:
    блок кода
```

if застосовують тоді, коли потрібно прийняти рішення (виконувати дію чи ні, залежно від умови)

ЗАУВАЖИМО!

- Однократне виконання тіла цикла називається **ітерацією.**
- Слово **for** пишеться маленькими літерами.
- Перший рядок цикла має завершуватися двокрапкою **(:).**
- Тіло цикла має відділятися **відступом.**

ЛІЧИЛЬНИК ЦИКЛУ ТА ФУНКЦІЯ RANGE()

Структура цикла **for** в мові Python:

```
for назва змінної in range(кількість повторень):
    блок кода
```

```
for i in range(10):
    print(i + 1, '-- Привіт')
```

```
1 -- Привіт
2 -- Привіт
3 -- Привіт
4 -- Привіт
5 -- Привіт
6 -- Привіт
7 -- Привіт
8 -- Привіт
9 -- Привіт
10 -- Привіт
```

```
for _ in range(5):
    print('Python - awesome!')
```

```
Python - awesome!
Python - awesome!
Python - awesome!
Python - awesome!
Python - awesome!
```

Для лічильників циклів зарезервовані літери **i, j, k**

Пам’ятаймо, що **права** межа циклу у Python ніколи **не включається у діапазон значень лічильника**

ПОПУЛЯРНІ СЦЕНАРІЇ ПРИ НАПИСАННІ ЦИКЛІВ

Перелічені сценарії - це класичні алгоритмічні шаблони.

1. **Підрахунок кількості**
2. **Підрахунок суми та добутку**
3. **Обмін значень змінних**
4. Сигнальні мітки
5. Пошук максимума / мінімума
6. Розширені оператори

ЦИКЛ FOR У STEM

Сценарій циклу	STEM-приклад	Коментар
Підрахунок кількості	Підрахунок кількості учнів, які виконали певне завдання в олімпіаді	**Math + Programming**: count += 1, умовні перевірки, статистика
Підрахунок суми/добутку	Розрахунок загальної ваги деталей у конструкції або сумарного балу на тесті	**Math**: арифметичні операції. **Engineering**: аналіз ресурсів
Обмін значень змінних	Перевірка коректності даних в таблиці (переставити значення місцями)	**Algorithmic thinking + Programming**: розуміння пам'яті
Сигнальні мітки	Переривання обчислень при досягненні умови: «як тільки знайшли мінімум - стоп»	**Engineering**: оптимізація процесів. **Logic**: дострокова зупинка
Пошук максимума/ мінімума	Знайти найкращий результат конкурсу або найвищу температуру	**Data Science + Math**: базова аналітика, алгоритм
Розширені оператори (range(step), enumerate, zip)	Створення звітів, парне оброблення значень, контроль індексу	**Technology**: "чистий" ефективний код. **Engineering**: структурування

СЦЕНАРІЙ. ПІДРАХУНОК КІЛЬКОСТІ

```
counter = 0
for _ in range(10):
    num = int(input())
    if num > 10:
        counter = counter + 1
print('Було введено', counter, 'чисел, більших 10.')
```

Для змінної лічильника зручно використовувати ім'я **counter** (або скорочено **cnt**).

2 кроки сценарію:

1. Створення змінної лічильника і надання їй початкового значення: **counter = 0**;
2. Збільшення змінної лічильника на 1: **counter = counter + 1.**

```
12
3
6
15
11
7
0
10
17
4
Було введено 4 чисел, більших 10.
```

СЦЕНАРІЙ. ПІДРАХУНОК СУМИ ТА ДОБУТКУ

```
total = 0
for _ in range(10):
    num = int(input())
    if num > 10:
        total = total + num
print('Сума чисел більших 10 дорівнює',  total)
```

```
Сума чисел більших 10 дорівнює 55
```

```
12
3
6
15
11
7
0
10
17
4
```

2 кроки сценарію:

1. Створення змінної суматора та надання їй початкового значення: **total = 0;**
2. Збільшення змінної суматора на потрібне число: **total = total + num**.

```
total = 0
for _ in range(10):
    num = int(input())
    total = total + num

average = total / 10
print('Середнє значення дорівнює', average)
```

```
Середнє значення дорівнює 8.5
```

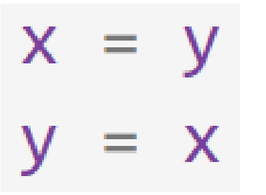
Для змінної суматора та мультиплікатора зручно використовувати ім'я **total**.

СЦЕНАРІЙ. ОБМІН ЗНАЧЕНЬ ЗМІННИХ

Програмісти-початківці напишуть такий код:

Помилковий код

```
x = y
y = x
```



```
temp = x
x = y
y = temp
```

Тимчасова змінна

У мові Python є більш простий спосіб обміну:

```
x, y = y, x
```



Аналогічно можна міняти значення
3-х та більше змінних

“РУЧНЕ” VS “РОЗУМНЕ” ПРОГРАМУВАННЯ

“Ручний” підхід	“Розумний” підхід
Повторюємо однакові дії вручну	Узагальнюємо за допомогою циклу
Код великий і важко читається	Код компактний і структурований
Складно змінювати межі обчислення	Можна легко змінити межі **(range)**
Ризик помилок через “copy - paste”	Менше ризиків, оскільки логіка централізована
Не масштабується	Легко масштабувати для будь-якого **n**

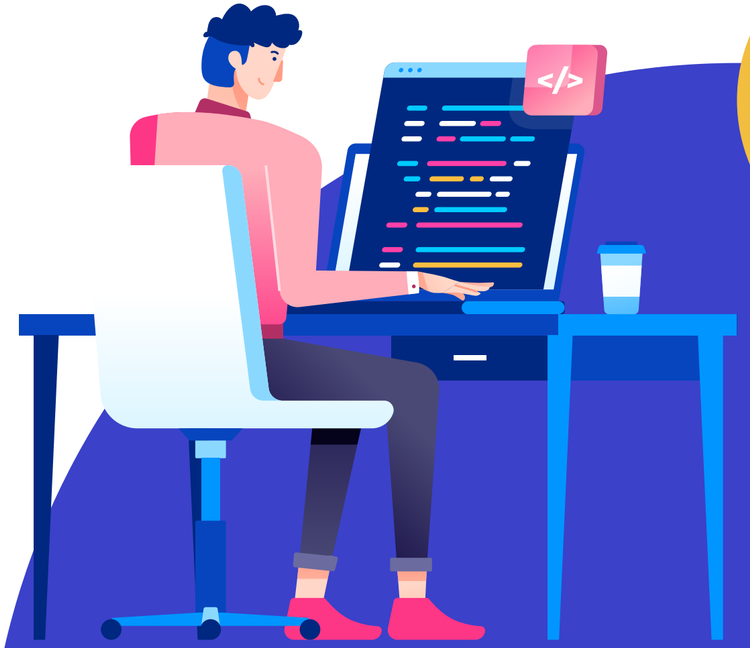
“РОЗУМНЕ” ПРОГРАМУВАННЯ = 

-ПІДХІД

Science	досліджуємо проблему
Technology	пишемо ефективний код
Engineering	шукаємо найкращу логіку
Mathematics	обґрунтовуємо результат

ЗАНЯТТЯ 9

ЦИКЛ FOR: ФУНКЦІЯ RANGE



ФУНКЦІЯ RANGE

Функція **range** у мові Python є вбудованою функцією, яка створює послідовність чисел всередині певного діапазону: повертає послідовність чисел, яка **починається з 0** (за замовчуванням), **збільшується на 1** (за замовчуванням) та **зупиняється** перед вказаним числом

ФУНКЦІЯ RANGE З ОДНИМ ПАРАМЕТРОМ

```
for i in range(10):
    print('Hello', i)
```

```
Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Hello 6
Hello 7
Hello 8
Hello 9
```

Змінна **i** приймає послідовно значення: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Функція **range (n)** генерує послідовність чисел від **0** до **n-1**, а цикл **for** послідовно перебирає цю послідовність.

ФУНКЦІЯ RANGE З ДВОМА ПАРАМЕТРАМИ

```
for i in range(3,7):
    print('Hello', i)
```

```
Hello 3
Hello 4
Hello 5
Hello 6
```

Змінна **i** приймає послідовно значення: 3, 4, 5, 6.

Функція **range (n, m)** генерує послідовність чисел від **n** до **m-1** з кроком **1,** а цикл **for** послідовно перебирає цю послідовність

ФУНКЦІЯ RANGE З ТРЬОМА ПАРАМЕТРАМИ

```
for i in range(3,10,2):
    print('Hello', i)
```

```
Hello 3
Hello 5
Hello 7
Hello 9
```

Змінна **i** приймає послідовно значення: 3, 5, 7, 9.

Функція **range (n, m, k)** генерує послідовність чисел від **n** до **m-1** з кроком **k**, а цикл **for** послідовно перебирає цю послідовність

ВІД’ЄМНИЙ КРОК

```
for i in range(5, 0, -1):
    print(i, end=' ')
print('Start!!!')
```

```
5 4 3 2 1 Start!!!
```

Змінна **i** набуває послідовно значення: 10, 9, 8, 7, 6, 5, 4.

У разі від’ємного кроку, маємо гарантувати, що **початок генерації (перший параметр)** більше ніж **кінець генерації (другий параметр)**

ПРИКЛАДИ ВИКОРИСТАННЯ ФУНКЦІЇ RANGE()

Оператор	Приклад
range(10)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
range(1, 10)	1, 2, 3, 4, 5, 6, 7, 8, 9
range(3, 7)	3, 4, 5, 6
range(7, 3)	порожня послідовність
range(2, 15, 3)	2, 5, 8, 11, 14
range(9, 2, -1)	9, 8, 7, 6, 5, 4, 3
range(3, 10, -2)	порожня послідовність

ПРИМІТКИ

1

1. Функція range() може приймати від 1 до 3-х параметрів: **range(stop), range(start, stop), range(start, stop, step)**
параметр **start** – початок перебору послідовності (включно)
параметр **stop** – кінець перебору послідовності (не включно)
параметр **step** – величина кроку

2

2. Функція **range()** може генерувати тільки цілі числа, включаючи і від'ємні.

3

3. Величина кроку не може дорівнювати нулю.

```
for i in range(1, 10, 0):
    print(i)
```

Отримаємо помилку! `ValueError: range() arg 3 must not be zero`

ПОПУЛЯРНІ СЦЕНАРІЇ ПРИ НАПИСАННІ ЦИКЛІВ

1. Підрахунок кількості
2. Підрахунок суми та добутку
3. Обмін значень змінних
4. **Сигнальні мітки**
5. **Пошук максимума / мінімума**
6. **Розширені оператори**

СИГНАЛЬНІ МІТКИ

Сигнальна мітка використовується для забезпечення взаємодії між різними частинами програми шляхом повідомлення про настання певної події або зміну стану в іншому її компоненті

Для сигнальної мітки використовується булева змінна, яка приймає два значення **True** або **False**.

Іноді сигнальні мітки називають **прапорцями**

СИГНАЛЬНІ МІТКИ

Програма зчитує число та визначає чи є воно простим

початкове значення сигнальної мітки

```
num = int(input())
flag = True
for i in range(2, num):
    if num % i == 0:
        flag = False
if flag == True:
    print('Число просте')
else:
    print('Число складене')
```

зміна значення сигнальної мітки

перевірка сигнальної мітки

ПОШУК МАКСИМУМА І МІНІМУМА

Пошук найбільшого (найменшого) значення в деякій послідовності чисел – *популярна задача в програмуванні*.

Ключем до пошуку найбільшого (найменшого) значення є використання додаткової змінної, в якій зберігається **поточне** найбільше (найменше) значення.

ПОШУК МАКСИМУМА

Програма зчитує 10 додатних чисел та знаходить серед них найбільше значення.

початкове значення приймаємо за -1

```
largest = -1
for i in range(10):
    num = int(input())
    if num > largest:
        largest = num
print(largest)
```

кожного разу, коли зчитуємо число більше **largest**, переприсвоюємо значення змінної **largest**

ПОШУК МАКСИМУМА

В якості початкового значення змінної приймаємо перший введений елемент:



в якості початкового значення приймаємо *перший введений елемент*

```
largest = int(input())
for i in range(9):
    num = int(input())
    if num > largest:
        largest = num
print(largest)
```



кожного разу, коли зчитуємо число більше за **largest**, переприсвоюємо значення змінної **largest**

ПОШУК МІНІМУМА

Для знаходження найменшого значення послідовності слід поміняти знак нерівності **(>)** на протилежний **(<).**

в якості початкового значення приймаємо *перший введений елемент*

```
smallest = int(input())
for i in range(9):
    num = int(input())
    if num < smallest:
        smallest = num
print(smallest)
```

кожного разу, коли зчитуємо число менше за **smallest**, переприсвоюємо значення змінної **smallest**

РОЗШИРЕНІ ОПЕРАТОРИ ПРИСВОЮВАННЯ

У програмах часто зустрічаються оператори присвоювання, де змінна, що стоїть зліва від знака **=**, одночасно використовується і у виразі праворуч.

```
counter = counter + 1
total = total + number
balance = balance - withdrawal
```



```
counter += 1
total += number
balance -= withdrawal
```


розширені оператори присвоювання

РОЗШИРЕНІ ОПЕРАТОРИ ПРИСВОЮВАННЯ

Оператор	Приклад	Еквівалент
+=	x += 5	x = x + 5
-=	x -= 2	x = x - 2
***=**	x *= 10	x = x * 10
/=	x /= 4	x = x / 4
//=	x //= 4	x = x // 4
%=	x %= 4	x = x % 4

ЗАНЯТТЯ 10

СТРУКТУРА ПОВТОРЕННЯ: ЦИКЛ WHILE



Скільки ітерацій зробить цикл?

```
for _ in range(1, 3, 12):
    print('Я працюю!')
```

А	1
Б	4
В	5
Г	жодної
Д	помилка виконання

Що побачимо на екрані після запуску програми?

```
x = 5
for num in range(3, 8):
    print(num, end=" ")
```

А	3 4 5 6
Б	5 5 5 5 5
В	3 4 5 6 7
Г	3 4 5 6 7 8
Д	5 5 5 5
Е	5 5 5 5 5 5

ЦИКЛИ

Перевага комп’ютерів перед людьми – здатність повторювати однакові дії багатократно.

В Python є **два** основні **різновиди циклів**:

- цикли, які повторюються певну кількість разів (**for, цикл з лічильником, counting loops**) (застосовується для заздалегідь відомої кількості повторень / ітерацій);
- цикли, які повторюються до того моменту, доки відбувається певна подія (**while, умовні цикли, conditional loops**).

ЦИКЛ WHILE

Цикл **while –** це керуюча конструкція в мові програмування Python, яка забезпечує багаторазове виконання блоку команд доти, поки задана умова залишається істинною

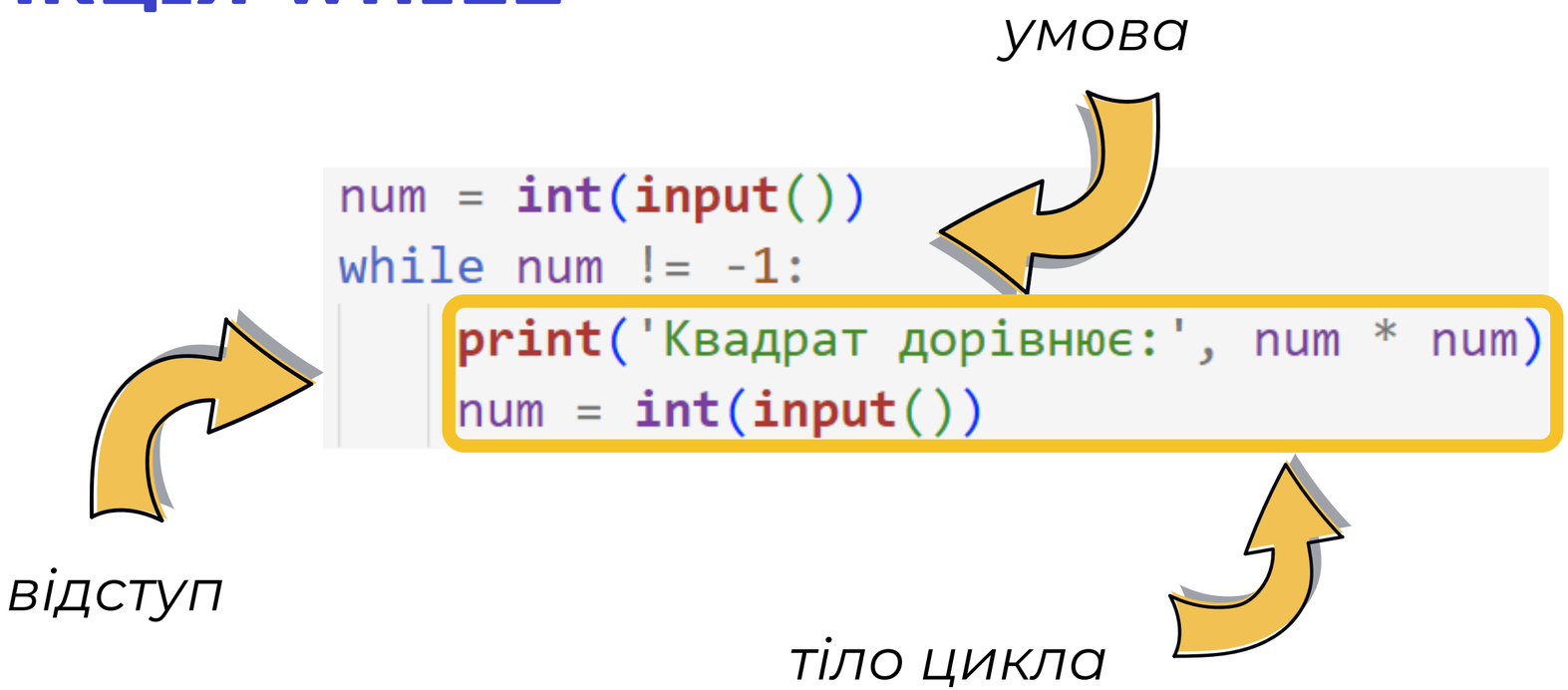
ЦИКЛ WHILE

Структура цикла **while** у Python:

```
while умова:
    блок кода
```

Двокрапка **(:)** в кінці рядка повідомляє Python, що далі знаходиться блок команд, який називається **тілом** цикла.

ФУНКЦІЯ WHILE

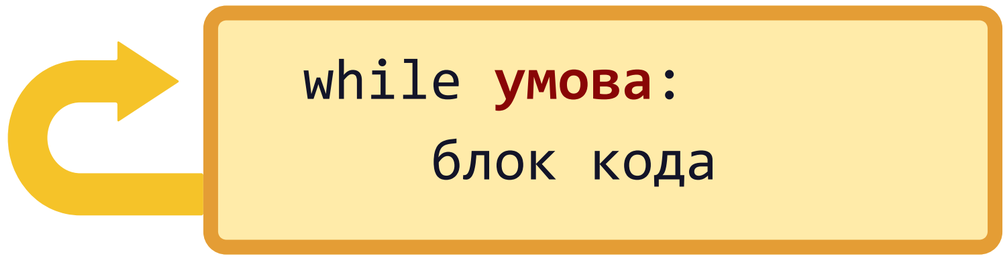


Важливі два моменти:

- попередня ініціалізація змінної **num;**
- зміна змінної **num** в тілі циклу **while.**

ЦИКЛ WHILE VS УМОВНИЙ ОПЕРАТОР IF

Структура цикла **while:**



```
while умова:
    блок кода
```

Структура умовного оператора **if:**



```
if умова:
    блок кода
```

ЦИКЛ WHILE VS ЦИКЛ FOR

Завжди можна замінити цикл **for** циклом **while:**

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



```
i = 0
while i < 101:
    print(i)
    i += 1
```

```
for i in range(0, 100, 3):
    print(i)
```



```
i = 0
while i < 100:
    print(i)
    i += 3
```



Не завжди вдається замінити цикл **while** на цикл **for**

НЕСКІНЧЕНИЙ ЦИКЛ

Цикл **while** повинен мати умову, за якої його виконання припиниться. Якщо такої умови немає або вона ніколи не стає хибною, цикл стає **нескінченним**.

```
i = 0
total = 0
while i < 10:
    total += i
print(total)
```

*змінна **i** не змінюється у тілі цикла та умова **i < 10** завжди істинна (True)*

Нескінчений цикл продовжує повторюватися доки, поки програма не буде *примусово перервана*

ЗАУВАЖЕННЯ

- Цикл **while** названий відповідно до принципу своєї роботи: він виконує дії доти, поки задана умова залишається **істинною**.
- Цикл **while** називається **циклом з передумовою**, оскільки виконанню тіла цикла передує перевірка умови.
- Цикл **while** може не виконатися жодного разу.

```
i = -1
while i > 0:
    print('Hello world!')
```

- Умова циклу **while**, як і в умовному операторі **if**, може містити логічні операції **or**, **and**, **not**.

ОБРОБКА ЦИФР ЧИСЛА

Нехай дано натуральне число **n**. Тоді:

- результатом операції **n %= 10** є остання цифра числа;
- результатом операції **n //= 10** є число з видаленою останньою цифрою.

ОБРОБКА ЦИФР ЧИСЛА

Програма зчитує число та обробляє його цифри:

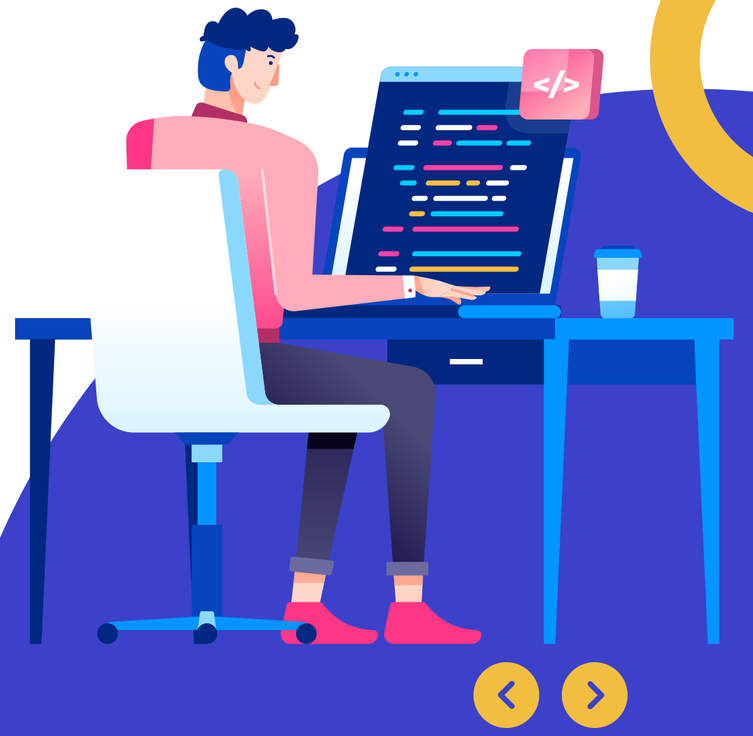
```
n = int(input())
while n != 0:                  # поки у числі є цифри
    last_digit = n % 10        # отримати останню цифру
    # код обробки останньої цифри
    n = n // 10                # видалити останню цифру з числа
```

В якості сценаріїв може бути:

- виведення цифр;
- знаходження суми або добутку цифр;
- знаходження найбільшої або найменшої цифри тощо.

ЗАНЯТТЯ 11

ОПЕРАТОРИ ПЕРЕРИВАННЯ



ОПЕРАТОР ПЕРЕРИВАННЯ BREAK

Оператор **break** перериває найближчий цикл **for** або **while.**

Наступна програма перевіряє, чи є введене число простим.

```
n = int(input())
pr = True
for i in range(2, n):
    if n % i == 0:
        pr = False
        break
if pr == True:
    print('Число просте')
else:
    print('Число складене')
```

*як тільки знайшли дільник, змінюємо значення сигнальної мітки та виходимо з циклу за допомогою оператора переривання **break***

Оператор **break** дозволяє оптимізувати роботу програми

ОПЕРАТОР ПЕРЕРИВАННЯ BREAK

Програма визначає, чи містить введене число цифру 7.

```
n = int(input())
pr = False
while n != 0:
    last_digit = n % 10
    if last_digit == 7:
        pr = True
        break
    n //= 10
if pr == True:
    print('Число', n, 'містить цифру 7')
else:
    print('Число', n, 'не містить цифру 7')
```

як тільки знайшли цифру 7, то виходимо з циклу за допомогою оператора переривання ***break***



Що покаже наведений фрагмент кода?

```
for i in range(10):
    print(i, end='*')
    if i > 6:
        break
```

А	0123456*
Б	0*1*2*3*4*5*6
В	0*1*2*3*4*5*6*
Г	01234567*
Д	0*1*2*3*4*5*6*7*

НЕСКІНЧЕННІ ЦИКЛИ

Якщо цикл не має можливості завершитися, то він називається **нескінченним циклом**.

Самий простий спосіб створити нескінченний цикл:

```
while True:
    print('Python awesome!')
```

```
Python awesome!
Python awesome!
.
.
.
Python awesome!
Python awesome!
```

```
while True:
    print('Python awesome!')
    if 5:
        break
```

```
Python awesome!
```

Оператор **break** дозволяє переривати нескінченні цикли

НЕСКІНЧЕННІ ЦИКЛИ

Іноді використання нескінченного циклу дозволяє зробити код **більш читабельним**.

Для цього доцільно завершувати цикл, ввівши умову всередині його тіла, а не через умову в заголовку.

```
while True:
    if умова 1: # умова для зупинки цикла
        break
    ...
    if умова 2: # ще одна умова для зупинки цикла
        break
    ...
```

ОПЕРАТОР ПРОПУСКА ІТЕРАЦІЙ CONTINUE

Оператор **continue** дозволяє перейти до наступної ітерації цикла **for** або **while** до завершення всіх команд в тілі цикла.

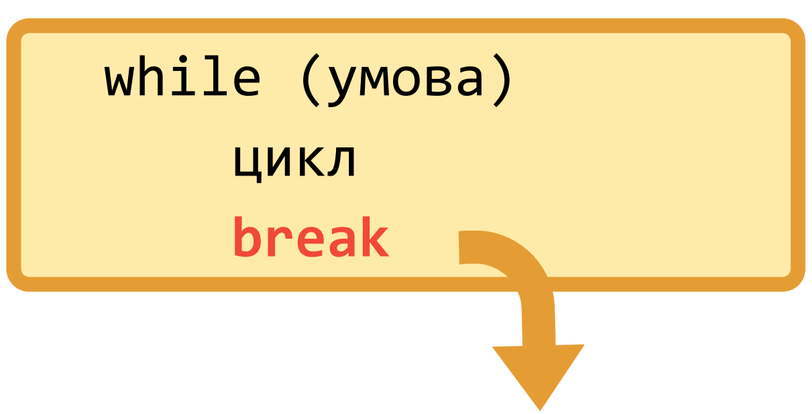
Програма виводить всі числа від 1 до 100, крім чисел 7, 17, 29 і 78.

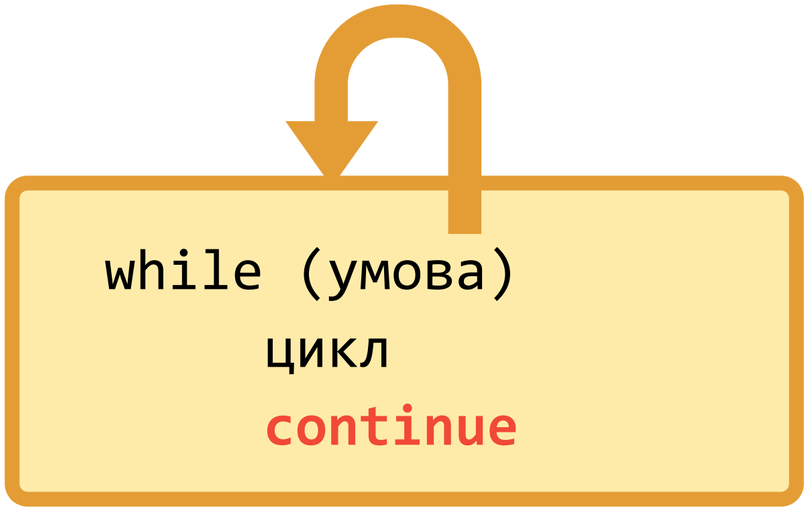
```
for i in range(1, 101):
    if i == 7 or i == 17 or i == 29 or i == 78:
        continue # переходимо на наступну ітерацію
    print(i, end=' ')
```

```
1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 25 26 27 28 30
 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54
 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 79
 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
```

ПРИМІТКИ

- Оператор **break** перериває виконання найближчого циклу, а не програми, тобто далі буде виконана команда, наступна одразу за **цим** циклом.
- Графічне подання операторів **break** і **continue** має вигляд:





БЛОК ELSE В ЦИКЛАХ

Python допускає необов’язковий блок **else** у циклах **while** та **for**. Синтаксис такої конструкції:

```
while умова:
    блок коду 1
else:
    блок коду 2
```

```
for i in range(10):
    блок коду 1
else:
    блок коду 2
```

Якщо в конструкції циклу відсутній блок **else**, то б**лок коду 2** виконується після завершення циклу незалежно від способу його завершення.

Якщо ж у структурі циклу передбачено блок **else**, то **блок коду 2** виконується лише за умови ***нормального (штатного) завершення*** *циклу, тобто без дострокового переривання за допомогою оператора* ***break***.

БЛОК ELSE В ЦИКЛАХ

Програма визначає, чи містить введене число цифру 7.

```
num = int(input())
n = num
while n != 0:
    last = n % 10
    if last == 7:
        print('Число', num, 'містить цифру 7')
        break
    n //= 10
else:
    print('Число', num, 'не містить цифру 7')
```

як тільки знайшли цифру 7, то виходимо з циклу за допомогою **break**

Порівняй з аналогічною програмою, розглянутою на початку заняття 11

ПРИМІТКИ

Примітка 1. Оператор **continue** жодним чином не впливає на виконання блоку **else** у циклах.

Примітка 2. Блок **else** у циклах часто застосовується для обробки відсутності елементів.

Примітка 3. Блок коду **else** у циклах зустрічається не так часто на практиці.

Однак якщо трапляється ситуація, в якій застосування **else** виправдане, його слід використовувати. Це може зробити ваш код більш зрозумілим і логічним

РЕВ’Ю КОДА

Рев’ю кода - перевірка вихідного кода програми з метою виявлення та виправлення **помилок та неточностей**, які залишилися непомітними при початковій розробці.

В процесі рев’ю кода можуть бути:

- виправлені фактичні помилки кода;
- підвищена продуктивність кода;
- покращена читабельність кода;
- усунуті помилки форматування кода.

Метою рев’ю кода є покращення якості програмного кода та вдосконалення навичок програміста.

ПОШИРЕНІ ПОМИЛКИ У ПРОГРАМАХ З ЦИКЛАМИ

1. Відсутність початкової ініціалізації змінної.
2. Неправильна початкова ініціалізація змінної.
3. Відсутність відступа.
4. Неправильні числові граничні значення (функція **range**).
5. Неправильні граничні порівняння.
6. Плутанина логічних операцій **or** та **and** тощо.

ЗАНЯТТЯ 12

ВКЛАДЕНІ ЦИКЛИ



Чи буде виконано блок **else** в наведеному нижче коді?

```
n = 0
while n < 10:
    n += 2
    print(n)
else:
    print('Цикл завершено')
```

А	Так
Б	Ні
В	За умови додавання **break**
Г	За умови додавання **continue**
Д	В циклі **while** не може бути **else**

ВКЛАДЕНІ ЦИКЛИ

Вкладений цикл - цикл, який розташований у середині іншого циклі.

Годинник є прикладом того, як працює вкладений цикл:

- секундна, хвилинна та годинникова стрілки обертаються навколо циферблата;
- годинникова стрілка зміщується всього на 1 крок для кожних 60 кроків хвилинної стрілки;
- хвилинна стрілка зміщується всього на 1 крок для кожних 60 кроків секундної стрілки;
- для кожного повного оберта годинникової стрілки (***12 кроків***), хвилинна стрілка робить 12*60 = 720 кроків, а секундна стрілка виконує 12*60*60 = 43 200 кроків

ВКЛАДЕНІ ЦИКЛИ

Розглянемо вкладені цикли, які частково моделюють роботу електронного годинника.

```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print(hours, ':', minutes, ':', seconds)
```

```
0 : 0 : 0
0 : 0 : 1
0 : 0 : 2
0 : 0 : 3
...
23 : 59 : 57
23 : 59 : 58
23 : 59 : 59
```

ВКЛАДЕНІ ЦИКЛИ

Приклад імітаційної моделі годинника підводить нас до деяких моментів, які мають відношення до вкладених циклів:

- вкладений цикл виконує всі свої ітерації для кожної окремої ітерації зовнішнього цикла;
- вкладені цикли завершують ітерації швидше, ніж зовнішні цикли;
- для того, щоб отримати загальну кількість ітерацій вкладеного цикла, слід перемножити кількість ітерацій всіх циклів.



Можна вкладати один в одний цикли як **for**, так і **while**

ОПЕРАТОРИ BREAK ТА CONTINUE У ВКЛАДЕНИХ ЦИКЛАХ

Оператор **break** виконує переривання найближчого цикла в якому він розташований.

Оператор continue здійснює перехід на наступну ітерацію **найближчого цикла**.

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

ОПЕРАТОРИ BREAK У ВКЛАДЕНИХ ЦИКЛАХ

Оператор **break** виконує переривання найближчого цикла в якому він розташований.

```
for i in range(3):
    for j in range(3):
        print(i, j)
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

*Приклад з оператором **break:***

```
for i in range(3):
    for j in range(3):
        if i == j:
            break
        print(i, j)
```

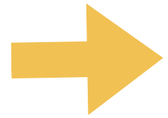
```
1 0
2 0
2 1
```

ОПЕРАТОРИ BREAK ТА CONTINUE У ВКЛАДЕНИХ ЦИКЛАХ

Оператор **continue** здійснює перехід на наступну ітерацію **найближчого цикла**.

Змінимо оператор переривання **break** на оператор **continue**:

```
for i in range(3):
    for j in range(3):
        if i == j:
            continue
        print(i, j)
```



```
0 1
0 2
1 0
1 2
2 0
2 1
```

Якщо слід виконати переривання зовнішнього цикла через виконання умови у внутрішньому, то варто виконати це через **сигнальну мітку.**

ВКЛАДЕНІ ЦИКЛИ ДЛЯ РОЗВ’ЯЗКУ РІВНЯНЬ

Знайти усі пари натуральних чисел (та їх кількість), які є розв’язком рівняння ***12x + 13y = 777.***

```
total = 0
for x in range(1, 65):
    for y in range(1, 60):
        if 12 * x + 13 * y == 777:
            total += 1
            print('x =', x, 'y =', y)
print("Загальна кількість натуральних розв'язків =", total)
```



```
x = 3 y = 57
x = 16 y = 45
x = 29 y = 33
x = 42 y = 21
x = 55 y = 9
Загальна кількість натуральних розв'язків = 5
```

ЗАНЯТТЯ 13

РЯДКОВИЙ ТИП ДАНИХ



Яку задачу розв’язує наведений нижче код?

```
n = int(input())
res = 1
i = 2
while i <= n:
    res *= i
    i += 1
print(res)
```

А	виводить суму чисел від 1 до n
Б	виводить факторіал числа n
В	виводить 2 в степені n
Г	виводить суму чисел від 2 до n

Яку задачу розв’язує наведений нижче код?

```
n = int(input())
i = 2
while n % i != 0:
    i += 1
print(i)
```

А	виводить кількість цифр у числі
Б	виводить найбільший дільник числа
В	виводить мінімальний дільник числа, який відрізняється від одиниці
Г	виводить скільки раз введене число ділиться на 2

РЯДКОВИЙ ТИП ДАНИХ

Рядок – це ***незмінна впорядкована*** послідовність символів, яка використовується для зберігання текстової інформації

РЯДКОВИЙ ТИП ДАНИХ

Створення рядка: використовуємо парні лапки **' '** або **" "**

```
s1 = 'Python'
s2 = "Python"
```

Створення рядка: можна використати **''' '''** або **""" """**

```
'''Ще один рядок'''
```

```
"""Ще один рядок"""
```

Зчитування рядка: використовуємо функцію **input()**

```
s = input()
num = int(input())
```

Порожній рядок: для створення пишемо **s = ''** або **s = "".**

Порожній рядок - *аналог* числа 0.

```
empty_str_1 = ''
empty_str_2 = ""
```

РЯДКОВИЙ ТИП ДАНИХ

```
print('Перший рядок лише з літер')
print('Це рядок з цифрами: 0123456789')
print('Тут є розділові знаки: !?.,:;')
print('Англійською: hello')
print('Китайською: 你好')
```

```
s = '''Hello,
world!
2025'''
print(s)
```

```
print("Hello,\nworld\n2025")
```

```
Hello,
world
2025
```

В такий спосіб створюють багаторядковий текст

Символ **\n** є службовим
(означає перенесення тексту на новий рядок).
Він складається з двох знаків: **зворотного слеша ** і **літери n**, але Python сприймає його як один символ

РЯДКОВИЙ ТИП ДАНИХ

Довжина рядка. Для визначення довжини рядка (кількості символів) застосовують вбудовану функцію **len().**

```
s = 'Hello'
n = len(s)
print(n)
```

довжина = 5

Перетворення числа в рядок - функція **str()**

```
num1 = 1777
num2 = 17.77
s1 = str(num1)
s2 = str(num2)
```

РЯДКОВИЙ ТИП ДАНИХ

Конкатенація та множення на число: оператори **+** та *****

```
print('AB' + 'cd')
print('A' + '7' + 'B')
print('Hi' * 4)
```

```
ABcd
A7B
HiHiHiHi
```

```
print('-' * 75)
```

```
---------------------------------------------------------------------------
```

Рядок можна множити на число, але не можна множити на рядок.

РЯДКОВИЙ ТИП ДАНИХ

Оператор входження in. За допомогою оператора **in** можна перевірити, чи входить один рядок до складу іншого.

```
s = 'All you need is love'
if 'love' in s:
    print('❤️')
else:
    print('💔')
```

*Рядок містить підрядок **'love'**, тому буде виведено символ ❤️*

У Python можна використовувати **emoji** 👍

EMOJI У PYTHON

```
emojis = '🐍🚀🔥'
print(emojis)
```

emoji — це пакет, який дозволяє перетворити вказаний ***shortcode*** у звичний вигляд емодзі (смайлик).

Для встановлення: `pip install emoji`

```
import emoji
print(emoji.emojize(":face_with_tears_of_joy:"))
print('🦝🦊🦝😺🐈')
```

ПОРІВНЯННЯ РЯДКІВ

Рядки рівні, якщо їх **довжина та символи** повністю збігатимуться. **Регістр** букв при даній операції також **важливий.**

Оператори порівняння рядків: **>** та **<**.
Python порівнює рядки посимвольно.

Символи порівнюються за їх кодами..

```
>>> "aaa" == "aaa"
True
>>> "AAA" == "aaa"
False
>>> "aaa" == "aa"
False
```

```
>>> 'a' > 'cde'
False
>>> 'abc' > 'bc'
False
>>> 'a' > 'A'
True
```

```
>>> 'abcd' > 'abc'
True
>>> ord('a')
97
>>> ord('A')
65
```

ІНДЕКСАЦІЯ

Індекс (від англ. «index») – це порядковий номер елемента у ***впорядкованій*** послідовності.

ІНДЕКСАЦІЯ

Для **звернення** до **певного символа** в рядку у мові Python використовують **квадратні дужки []**, де вказується ***індекс (номер)*** потрібного символа.



Операція звернення за індексом називається **індексацією** рядка.

ІНДЕКСАЦІЯ

s = 'Python'

Вираз	Результат	Пояснення
s[0]	**P**	1-й символ рядка
s[1]	**y**	2-й символ рядка
s[2]	**t**	3-й символ рядка
s[3]	**h**	4-й символ рядка
s[4]	**0**	5-й символ рядка
s[5]	**n**	6-й символ рядка

Перший символ рядка **s[0]**, а не s[1]. У мові Python індексація **починається з 0.**

ІНДЕКСАЦІЯ

s = 'Python'

Вираз	Результат	Пояснення
s[-6]	**P**	1-й символ рядка
s[-5]	**y**	2-й символ рядка
s[-4]	**t**	3-й символ рядка
s[-3]	**h**	4-й символ рядка
s[-2]	**0**	5-й символ рядка
s[-1]	**n**	6-й символ рядка

Якщо перший символ рядка має індекс **0**, то останньому символу присвоюється індекс **-1.**

ІНДЕКСАЦІЯ

s = 'Python'

Додатні індекси	0	1	2	3	4	5
Рядок	**P**	**y**	**t**	**h**	**o**	**n**
Від’ємні індекси	-6	-5	-4	-3	-2	-1

При зверненні до неіснуючого індексу в рядку виникає помилка:

IndexError: string index out of range

ІТЕРУВАННЯ РЯДКІВ

Ітерування за індексами:

```
s = 'abcdef'
for i in range(len(s)):
    print(s[i])
```

Ітерування за символами:

```
s = 'abcdef'
for c in s:
    print(c)
```

```
a
b
c
d
e
f
```

Зверніть увагу на назву змінних цикла:

- в першому циклі ми використовуємо ім’я **i**, що відповідає стандартній ідеології найменування змінних цикла;
- у другому циклі назвали змінну літерою **c** - перша літера слова char (символ).

ЗРІЗИ

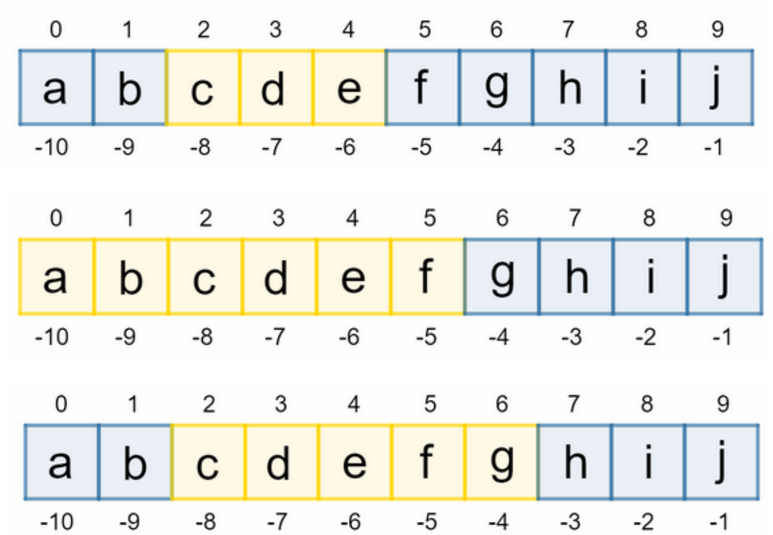
Зріз (від англійського слова «slice») - це механізм роботи з послідовностями, який дає змогу отримати підпослідовність елементів (зокрема, кілька символів рядка) за один виклик.
При цьому **вихідний** рядок **не змінює** свого стану.

ЗРІЗИ `рядок[початок зрізу:кінець зрізу]`

Індекси	0	1	2	3	4	5	6	7	8	9
Рядок	a	b	c	d	e	f	g	h	i	j
Індекси	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
s = 'abcdefghij'
print(s[2:5])
print(s[0:6])
print(s[2:7])
```

```
cde
abcdef
cdefg
```



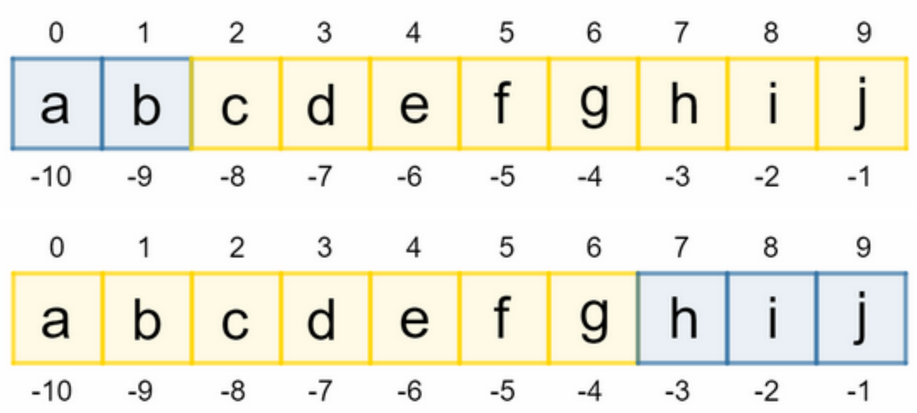
У зрізі **s[x:y] перше число** - це місце, де починається зріз (**включно**), а **друге** - це місце, де закінчується зріз (**невключно**).

ЗРІЗИ ДО КІНЦЯ, ВІД ПОЧАТКУ

- Якщо у виразі зрізу **s[x:]** не вказано другий параметр, то підпослідовність формується від позиції x до кінця рядка.
- Якщо ж у виразі **s[:y]** опущено перший параметр, то зріз охоплює елементи від початку рядка до позиції **y** (не включаючи її).

```
s = 'abcdefghij'
print(s[2:])
print(s[:7])
```

```
cdefghij
abcdefg
```



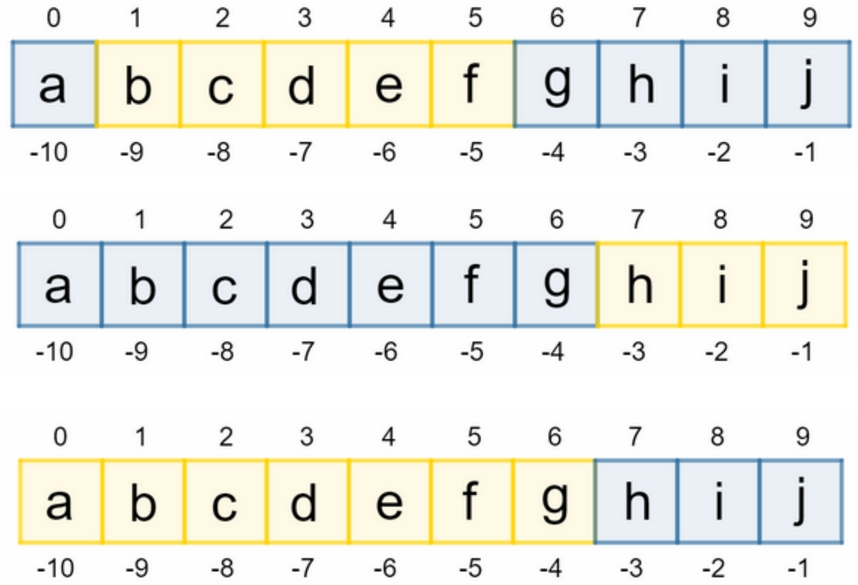
Зріз **s[:]** співпадає з самим рядком **s**

ВІД'ЄМНІ ІНДЕКСИ У ЗРІЗІ

Можна використовувати від'ємні індекси для створення зрізів. При використанні від'ємних індексів ***перший параметр зріза має бути менше за другий, або має бути пропущений.***

```
print(s[-9:-4])
print(s[-3:])
print(s[:-3])
```

```
bcdef
hij
abcdefg
```



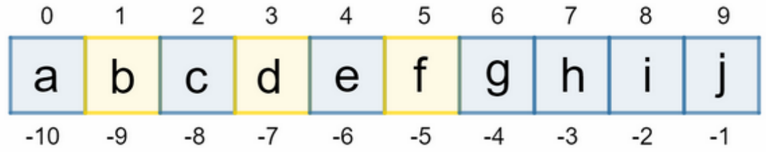
Від'ємні індекси рядка починаються з **-1** та відраховуються до досягнення початку рядка

КРОК ЗРІЗА

Можна передати у зріз третій необов’язковий параметр, який відповідає за крок зріза.

```
s = 'abcdefghij'
print(s[1:7:2])
```

→ bdf

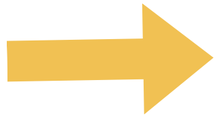


Індекси	0	1	2	3	4	5	6	7	8	9
Рядок	a	**b**	c	**d**	e	**f**	g	h	i	j
Індекси	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

ВІД’ЄМНИЙ КРОК ЗРІЗА

Якщо в якості крока вказати від’ємне число, то символи будуть йти в звороному порядку.

```
s = 'abcdefghij'
print(s[::-1])
print(s[::-2])
```



```
jihgfedcba
jhfdb
```

Індекси	0	1	2	3	4	5	6	7	8	9
Рядок	a	b	c	d	e	f	g	h	i	j
Індекси	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

ПІДСУМУЄМО

s = 'abcdefghij'

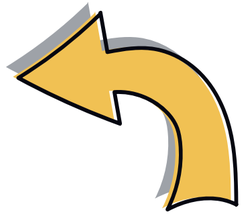
Код	Результат	Пояснення
s [2 : 5]	**cde**	рядок, що складається з символів з індексами 2, 3, 4
s [: 5]	**abcde**	перші п’ять символів рядка
s [5 :]	**fghij**	рядок, що складається з символів з індексами від 5 і до кінця
s [-2 :]	**ij**	останні два символа рядка
s [:]	**abcdefghij**	увесь рядок
s [1 : 7 : 2]	**bdf**	рядок, що складається з кожного другого символа з індексом від 1 до 6
s [1 : 7 : 2]	**jihgfedcba**	рядок у зворотному порядку, т.я. крок від’ємний

ЗМІНА СИМВОЛІВ РЯДКА ЗА ІНДЕКСОМ

Якщо значення кроку у зрізі є від’ємним, то елементи послідовності обираються у зворотному порядку.

```
s = 'abcdefghij'
```

```
s[4] = 'X'
```



Помилка!
У Python ***рядки є незмінними***

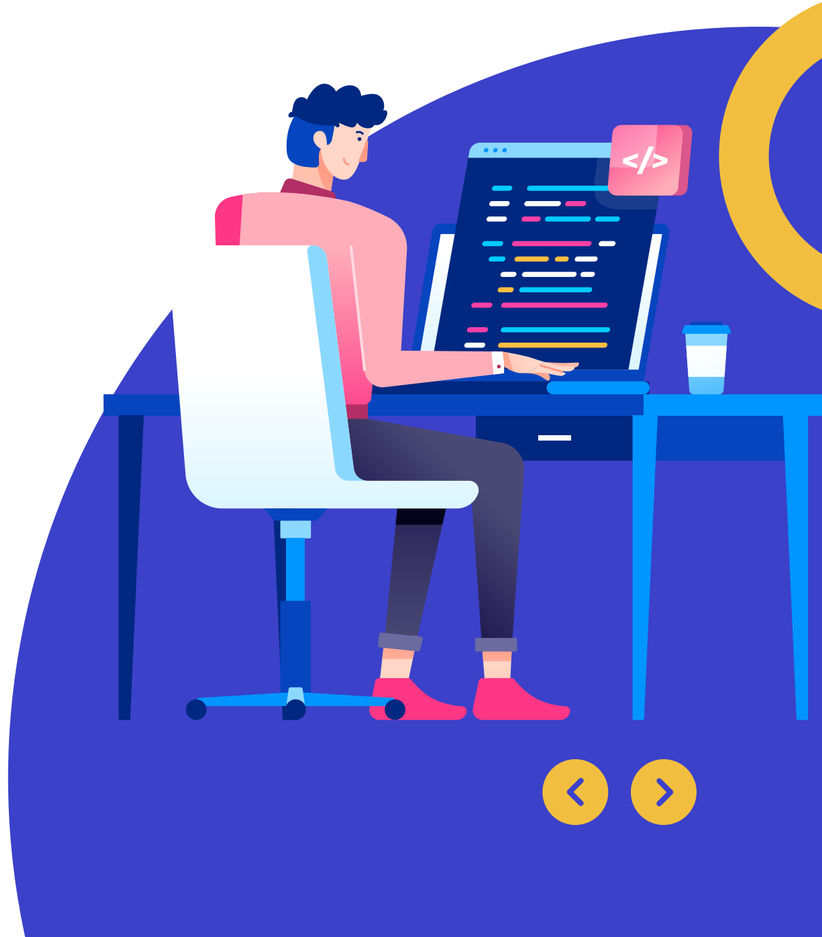
```
s = 'abcdefghij'
s = s[:4] + 'X' + s[5:]
```



Вирішення:
Проблема вирішується ***створенням нового рядка***

ЗАНЯТТЯ 14

ОСНОВИ РОБОТИ ЗІ СПИСКАМИ



Що покаже наведений нижче код?

```
print(min('9', '10', '11') + max('9', '10', '11'))
```

А	20
Б	19
В	109
Г	911
Д	21
Е	виникне помилка

Є рядок name

```
name = 'Python'
```

Як правильно отримати символ 'y' ?

А	name{0}
Б	name[1]
В	name{1}
Г	name[0]
Д	name([1])
Є	name(1)

ВВЕДЕННЯ У СПИСКИ

Структура даних (data structure) –
програмна одиниця, що дозволяє зберігати та обробляти
множину однотипних та/або логічно пов'язаних даних

ВВЕДЕННЯ У СПИСКИ

Визначення 1. **Список –** це структура даних, яка дозволяє зберігати та обробляти множину однотипних та / або логічно пов’язаних даних

Визначення 2. **Список –** це послідовність елементів, пронумерованих від 0, як символи у рядку.



У більшості мов програмування списки називаються **масивами**

СТВОРЕННЯ СПИСКА

Для створення списку його елементи записують **через кому**, розміщуючи їх **у квадратних дужках**:

```
numbers = [2, 4, 6]
languages = ['Python', 'C#', 'C++', 'Java']
```

Список **numbers** складається з 3 елементів

```
numbers[0] = 2
numbers[1] = 4
numbers[2] = 6
```

Список **languages** складається з 4 елементів

```
languages[0] = 'Python'
languages[1] = 'C#'
languages[2] = 'C++'
languages[3] = 'Java'
```

Значення, які розташовані у квадратних дужках та виокремлені комами, називаються **елементами списка**

СТВОРЕННЯ СПИСКА

Список може містити значення даних різних типів:

```
info = ['Timur', 1992, 61.5]
```

Список **info** містить рядкове значення, ціле число та число з плаваючою крапкою

```
info[0] = 'Timur'
info[1] = 1992
info[2] = 61.5
```

Зазвичай елементи списка мають однаковий тип даних

ПОРОЖНІЙ СПИСОК

Створити порожній список можна ***2 способами.*** Використавши:

1) порожні квадратні дужки **[];**
2) вбудовану функцію **list().**

```
mylist = []      # порожній список
mylist = list()  # порожній список
```

ВИВЕДЕННЯ СПИСКА

Для виведення всього списку на екран можна скористатися функцією **print()**:

```
numbers = [2, 4, 6, 8, 10]
languages = ['Python', 'C#', 'C++', 'Java']
print(numbers)
print(languages)
```

Функція **print()** виводить на екран елементи списка в квадратних дужках, що розділені комами:

```
[2, 4, 6, 8, 10]
['Python', 'C#', 'C++', 'Java']
```

ВБУДОВАНА ФУНКЦІЯ LIST()

Мова Python має вбудовану функцію **list()**, яка крім створення порожнього списку може перетворити певні типи об’єктів у списки:

```
numbers = list(range(5))
even_numbers = list(range(0, 10, 2))
chars = list('abcde')
```

```
numbers =  [0, 1, 2, 3, 4]
even_numbers = [0, 2, 4, 6, 8]
chars =  ['a', 'b', 'c', 'd', 'e']
```

РОБОТА ЗІ СПИСКАМИ

Робота зі списками має багато спільного з роботою з рядками, адже обидві структури даних складаються з окремих елементів. Водночас між ними існує принципова відмінність:

елементи списку можуть бути довільного типу,
тоді як ***елементи рядк****а* завжди є символами

Більшість функціоналу, що доступно під час роботи з рядками, також доступно і при роботі зі списками.

ОПЕРАЦІЇ НАД СПИСКАМИ

1. Знаходження **довжини списка**: вбудована функція **len()**

> ❗ Вкладені списки вважаються одним елементом, а не рахуються за кількістю вкладених значень

```
lst = [5, 6, 7]
print(len(lst))
```

2. **Додавання (конкатенація)** списків: **оператор +**

> ❗ НЕ можна складати список з числом

```
a = [1, 2, 3]
a = ['hi'] + a
print(a)
```

```
['hi', 1, 2, 3]
```

3. **Дублювання** списку: списки можна множити на ціле число, що призводить до **повторення елементів**

> ❗ При множенні списка на нуль результатом буде порожній список

4. **Перевірка входження елемента у список:** оператор **in.**

Результат: **True / False** – елемент знайдено / не знайдено у списку

> ❗ Пошук відбувається лише за елементами

ОПЕРАЦІЇ НАД СПИСКАМИ

5 Знаходження **мінімального та максимального елемента** у списку: **min() та max()**



Функція працює тільки зі списками, що повністю складаються з чисел. Якщо у списку є різні типи даних, ***виникне помилка***

```
a = [2, 5, 8, 9]
print(max(a))
print(min([4,2,7]))
```

6 **Додавання всіх** елементів списка: функція **sum()**



Функція працює тільки з числами.
Якщо у списку є рядки, ***виникне помилка***

```
a = [2, 5, 9, 8]
print(sum(a) / len(a))
```

7 **Сортування** списка: функція **sorted().**
За замовчуванням - за зростанням.

```
a = [4, 2, 9, 1]
print(sorted(a, reverse=True))
# [9, 4, 2, 1]
```

8 **Порівняння** списків: як і рядки списки порівнюються за елементами (знаки <, >, ==). Результат **True** або **False**

ЯК ЗЧИТАТИ СПИСОК ЧИСЕЛ ВІД КОРИСТУВАЧА?

- Метод **.split()** розбиває введений рядок на окремі елементи за пробілами та повертає список рядків

```
data = input("Введіть числа у рядок через пробіл: ").split()
print(data)
```

```
Введіть числа у рядок через пробіл: 2 5 67 43
['2', '5', '67', '43']
```

- Метод . map() перетворює введений рядок у **список чисел:**

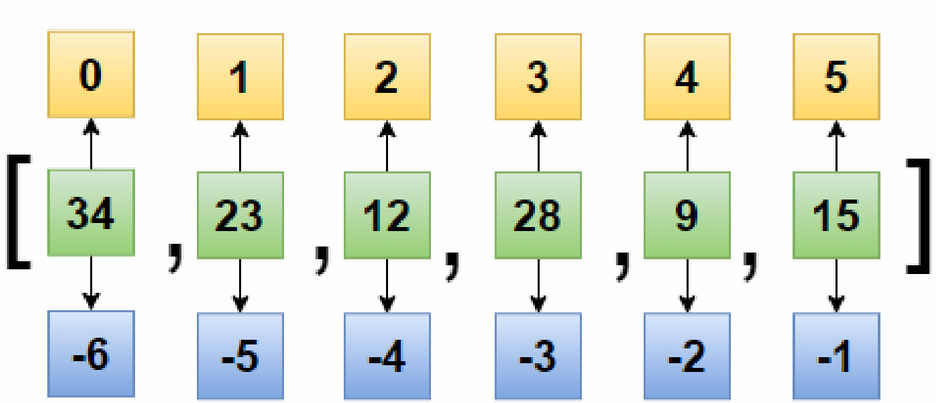
list(map(int, input().split()))

```
my_list = list(map(int, input('Введіть значення: ').split()))
print(f'Створений список: {my_list}')
```

```
Введіть значення: 2 5 67 43
Створений список: [2, 5, 67, 43]
```

ІНДЕКСАЦІЯ

Для **доступу** до **конкретного елемента** в списку Python використовують **квадратні дужки []**, у яких зазначається ***індекс (порядковий номер)*** цього елемента.



Операція звернення за індексом називається **індексацією списка.** У Python **індексація починається з 0.**

При зверненні до неіснуючого індексу в рядку виникає помилка: **IndexError: list index out of range**

ЗРІЗИ

Зріз від початку до кінця: **список[start:stop]**

- у зрізі **lst [x :] –** зріз береться до кінця рядка.
- у зрізі **s [: y] –** зріз береться від початку рядка.

Зріз **lst [:]** повертає копію вихідного списку

Від’ємні індекси можна використовувати для створення зрізів.

Від’ємні індекси списка починаються з **-1** та відраховуються до досягнення початку списка

Крок зріза: **список[start:stop:step]**

ЗМІННІСТЬ СПИСКІВ

Список, на відміну від рядка, є змінним об'єктом.

```
notes = ['До', 'Ре', 'Мі', 'Фа', 'Соль', 'Ля', 'Сі']
print(notes)
notes[1] = 'Ба'
print(notes)
notes[3:5] = [100, 'Ми']
print(notes)
del notes[5]
print(notes)
notes[1:5] = ['Ось', 'Так']
print(notes)
```

```
['До', 'Ре', 'Мі', 'Фа', 'Соль', 'Ля', 'Сі']
['До', 'Ба', 'Мі', 'Фа', 'Соль', 'Ля', 'Сі']
['До', 'Ба', 'Мі', 100, 'Ми', 'Ля', 'Сі']
['До', 'Ба', 'Мі', 100, 'Ми', 'Сі']
['До', 'Ось', 'Так', 'Сі']
```

```
a = [1, 2, 3]
b = a
b[0] = 10
print(a)
print(b)
```

```
[10, 2, 3]
[10, 2, 3]
```

ПОДІБНІСТЬ СПИСКІВ ТА РЯДКІВ

- функція **len()** - для підрахунку довжини списка;
- оператор входження **in;**
- індексація (звернення за індексом);
- зрізи (підсписки вихідних списків);
- операція конкатенації та множення на число;
- функції **min()** та **max()**.

ВІДМІННІСТЬ СПИСКІВ ТА РЯДКІВ

- **списки змінні**, іноді їх називають ***мутуючими колекціями*** (рядки незмінні);
- функція **sum(),** яка знаходить суму елементів списка, відсутня у рядків.

ВИВЕДЕННЯ ЕЛЕМЕНТІВ СПИСКУ

Ітерування за індексами:

```
numbers = [0, 1, 2, 3, 4, 5]
for i in range(len(numbers)):
    print(numbers[i])
```

Ітерування за елементами:

```
numbers = [0, 1, 2, 3, 4, 5]
for num in numbers:
    print(num, end=' ')
```

Виведення за допомогою розпаковки списку:

```
numbers = [0, 1, 2, 3, 4, 5]
print(*numbers)
```

```
0 1 2 3 4 5
```

Оскільки рядок є послідовністю символів (аналогічно до того, як список є послідовністю елементів), до нього можна застосовувати операцію **розпакування** за тими самими принципами, що й до списків.

```
s = 'Python'
print(*s)
print()
print(*s, sep='\n')
```

ДЛЯ НОТАТОК

Миколаївський обласний інститут післядипломної педагогічної освіти
Кафедра теорії й методики природничо-математичної освіти та інформаційних технологій

Старт у світ кодування: Python для початківців

Літературний редактор:
Н. А. Махровська, Г. С. Погромська

Комп’ютерний набір, технічна верстка:
Г. С. Погромська, Н. А. Махровська

Папір офсетний.
Друк на різографі
Формат 60x84/16
Умовно-друкованих арк. – 6,75
Обліково-видавничих арк. – 6,3
Гарнітура шкільна

Тираж 100
Замовлення №

Адреса редакції:
вул. Адміральська, 4-а,
м. Миколаїв, 54001
Тел./факс 37 85 89

http://www.moippo.mk.ua
e-mail: moippo@moippo.mk.ua