ЛЕКЦІЯ 1 · PYTHON

Python: синтаксис, дані та алгоритми

Числа, рядки, колекції, умови й цикли

Дисципліна «Мова програмування Python»
Викладач: Голенко Максим Юрійович

План лекції 1

01 **Python і синтаксис**

Сфери застосування, запуск, правила запису

02 **Числа, введення та текст**

Змінні, обчислення, форматування, рядки

03 **Колекції**

Списки, кортежі, множини та словники

04 **Умови й цикли**

Рішення, повторення та обробка записів

Python і сфери застосування

З’ясуємо, які задачі розв’язують за допомогою Python.

Далі: запуск програми та правила синтаксису.

Що таке Python

Python — мова загального призначення для написання читабельних програм.

Синтаксис

Відступи визначають блоки. Код близький до послідовності дій.

Типізація

Тип має об’єкт. Ім’я може посилатися на різні значення.

Екосистема

Стандартна бібліотека й сторонні пакети дають готові засоби.

Приклад обчислення: total = price * quantity

Де використовується Python

Python застосовують у різних сферах: від невеликих скриптів до систем аналізу даних.

Автоматизація

Файли, звіти, перевірки, обробка журналів. Також інструменти кібербезпеки.

Веб і застосунки

API та серверна логіка; Django, Flask. GUI — tkinter та інші засоби.

Дані та наука

Аналіз таблиць, графіки, моделювання, машинне навчання.

Обираємо інструмент за задачею: автоматизація / веб / дані / ШІ.

Python у штучному інтелекті

Python допомагає підготувати дані, навчити модель і використати її прогноз.

Машинне навчання

Класифікація та прогнозування. Приклад інструмента: scikit-learn.

Нейронні мережі

Робота з текстом, зображеннями, звуком. PyTorch і TensorFlow.

Готові моделі

Використання навченої моделі локально або через API.

Дані → **Підготовка** → **Модель** → **Прогноз**

Навчання моделі та використання готової моделі — різні етапи.

Від скрипту до проєкту

Масштаб програми зростає разом із кількістю задач і частин коду.

Скрипт

Один файл для конкретної дії: обчислити або перетворити дані.

Модуль і пакет

Пов’язані функції та класи розділяють на файли й каталоги.

Застосунок

Кілька частин, залежності, збереження даних і тести.

Наша задача: отримати оцінки → обробити → сформувати звіт.

Що входить у три лекції

Три лекції формують основу для подальшого практичного навчання.

Лекція 1

Синтаксис, типи, текст, колекції, умови й цикли.

Лекція 2

Функції, перетворення даних, винятки, журнали й тести.

Лекція 3

Модулі, файли, ООП та огляд прикладних бібліотек.

Результат: невелика програма обробки записів і розуміння її коду.

Виконання програми та синтаксис

Запустимо програму й розглянемо правила запису коду.

Від задач переходимо до першої виконуваної інструкції.

Інтерпретатор і способи запуску

REPL

Команди по одній

```
>>> 2 + 3
5
```

Інтерпретатор одразу показує значення виразу. Символи >>> є запрошенням до введення.

Файл .py

Збережена програма

```
# hello.py
print(2 + 3)
# Запуск: python hello.py
```

У файлі результат виводимо через print(). Запуск виконує інструкції збереженої програми.

У файл .py не записуємо >>>. IDE також запускає інтерпретатор.

Перша програма

Програма обчислює вартість товарів і показує результат у консолі.

PYTHON · ПРИКЛАД

```python
print("Вітаємо!")
price = 120
quantity = 3
total = price * quantity
print("Вартість:", total)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

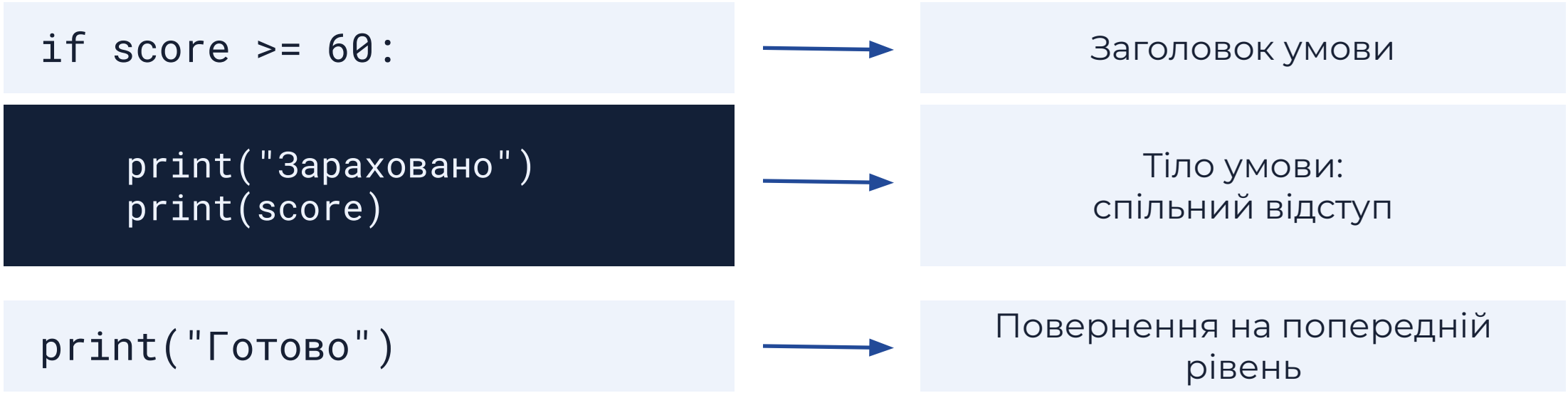
```
Вітаємо!
Вартість: 360
```

Пояснення

Інструкції виконуються згори вниз. Спочатку друкується привітання, потім добуток 120 × 3 зберігається в total. Останній print показує підпис і обчислену вартість 360.

Відступи визначають блоки

Двокрапка відкриває блок; однаковий відступ означає один рівень.



Звичайний стиль — 4 пробіли. Не змішуйте табуляцію з пробілами.

Відступи та блоки

Програма перевіряє оцінку та показує, які інструкції належать до умовного блоку.

PYTHON · ПРИКЛАД

```python
score = 75
if score >= 60:
    print("Зараховано")
    print("Результат:", score)
print("Перевірку завершено")
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Зараховано
Результат: 75
Перевірку завершено
```

Пояснення

За оцінки 75 умова істинна, тому виконуються обидва рядки з відступом. Останній print стоїть поза блоком if і виконується незалежно від результату перевірки.

Імена, коментарі та стиль

Приклад демонструє зрозумілі імена змінних і коментар до обчислення вартості.

PYTHON · ПРИКЛАД

```python
# Вартість одного товару
unit_price = 120
quantity = 3
total_price = unit_price * quantity
print(total_price)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
360
```

Пояснення

Коментар після # не виконується. Ім’я може містити літери, цифри й _, але не починається з цифри. Регістр важливий. Не використовуйте ключові слова та імена вбудованих засобів, як-от list, str чи input.

Як виконується Python

У CPython код проходить кілька етапів перед отриманням результату.

Код .py

Текст програми з інструкціями та відступами.

Байткод

Інтерпретатор компілює код у внутрішнє подання.

Виконання

Віртуальна машина виконує інструкції байткоду.



CPython: байткод виконує інтерпретатор.

Змінні, числа й оператори

Навчимося зберігати значення та виконувати обчислення.

Синтаксис зрозумілий — тепер працюємо з даними.

Ім’я, об’єкт і тип

Ім’я зберігає посилання на об’єкт. Об’єкт має значення та тип, який визначає допустимі операції.

Перше присвоєння

`value → 10  (int)`

value = 10 пов’язує ім’я з цілим числом.

Нове присвоєння

`value → "10" (str)`

value = "10" пов’язує те саме ім’я з іншим об’єктом — текстом.

Число 10 не перетворюється на рядок. Змінюється посилання імені value. Зміна вмісту об’єкта — інша дія, яку дозволяють лише змінні типи.

Незмінні: int, float, bool, str, tuple. Змінні: list, set, dict.

Імена посилаються на об’єкти

Програма показує, як присвоєння змінює зв’язок імені з об’єктом.

PYTHON · ПРИКЛАД

```python
value = 10
print(value, type(value))
value = "10"
print(value, type(value))
other = value
print(other)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
10 <class 'int'>
10 <class 'str'>
10
```

Пояснення

Після першого присвоєння value посилається на int, після другого — на str. Друк значень схожий, але type показує різні типи. other отримує посилання на той самий рядок.

Базові типи

PYTHON · ПРИКЛАД

```python
count = 5           # int
price = 12.5        # float
passed = True       # bool
name = "Олена"      # str
result = None       # немає значення
print(type(price).__name__)
print(isinstance(count, int))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
float
True
```

Пояснення

int — цілі числа, float — дробові, bool — True або False, str — текст. None позначає відсутність значення. type(price).__name__ дає назву типу, а isinstance(count, int) перевіряє належність до int.

Арифметика: оператори та правила

Запис	Призначення	Нюанс
`a + b; a - b`	Сума; різниця	Типи операндів мають бути сумісні
`a * b; a / b`	Добуток; ділення	/ для цілих повертає float
`a // b; a % b`	Частка вниз; остача	b не може дорівнювати 0
`a ** b`	Степінь	2 ** 3 = 8
`(a + b) * c`	Групування	Дужки змінюють порядок дій
`a += b`	Скорочене присвоєння	Для змінних об’єктів є нюанси

Для -17 // 5 результат -4: округлення вниз, а не відкидання дробу.

Арифметичні оператори

PYTHON · ПРИКЛАД

```python
a = 17
b = 5
print(a / b)
print(a // b)
print(a % b)
print(2 ** 3)
print(2 + 3 * 4)
print(-17 // 5)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
3.4
3
2
8
14
-4
```

Пояснення

17 / 5 дає 3.4, а 17 // 5 — цілу частку 3. Остача дорівнює 2. Множення виконується перед додаванням. Для від'ємної частки // округлює вниз: −3.4 стає −4.

Присвоєння та розпакування: запис

Присвоєння пов’язує ім’я з результатом виразу. Спочатку обчислюють праву частину.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
ім’я = вираз

ім’я_1, ім’я_2 = значення_1, значення_2

ім’я += приріст
```

Ім’я та =

Ліворуч — ім’я, праворуч — вираз. = виконує присвоєння.

Розпакування

Імена отримують відповідні значення. Кількість має збігатися.

Скорочений запис

Для числа ім’я += приріст додає приріст до значення імені.

Обмін a, b = b, a можливий без допоміжного імені: праву частину обчислюють першою.

Присвоєння й розпакування

Приклад демонструє обмін значеннями, розпакування та скорочене присвоєння.

PYTHON · ПРИКЛАД

```python
a, b = 2, 7
a, b = b, a
print(a, b)
score = 60
score += 5
print(score)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
7 2
65
```

Пояснення

Спочатку a і b отримують 2 та 7. Під час обміну права частина використовує їхні попередні значення, тому результат — 7 і 2. Для числа score += 5 збільшує оцінку до 65.

Перетворення типів: призначення і запис

Перетворення створює значення потрібного типу. Його застосовують, коли вхідні дані й потрібна операція мають різні типи.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
int(значення)
float(значення)
str(значення)
bool(значення)
```

int / float

int — ціле, float — дробове число. Текст має бути допустимим.

str

str створює текстове подання об’єкта.

bool

0, None та порожні колекції дають False.

int(12.5) відкидає дробову частину. int("12.5") дає помилку — це дробовий текст.

Перетворення типів

Приклад показує перетворення тексту на числа та перевірку істинності рядків.

PYTHON · ПРИКЛАД

```python
text = "12"
number = int(text)
price = float("12.5")
print(number + 3)
print(str(number) + " грн")
print(bool(""), bool("0"))
# int("12.5") → ValueError
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
15
12 грн
False True
```

Пояснення

int перетворює текст «12» на число для додавання. str потрібний для поєднання з текстом « грн». Порожній рядок дає False, а непорожній «0» — True. Дробовий текст не приймається int.

Числові функції: аргументи й результат

Вбудовані числові функції виконують поширені обчислення. Аргументи записують у круглих дужках.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
abs(число)
round(число, кількість_знаків)
min(значення_1, значення_2, ...)
max(значення_1, значення_2, ...)
pow(основа, показник)
```

abs / round

abs — модуль, round — округлення. Число знаків необов’язкове.

min / max

Обирають крайні значення серед аргументів або в колекції.

pow

pow(основа, показник) підносить основу до степеня показника.

Функція повертає результат. Для його показу в консолі використовують print().

Корисні числові засоби

Програма знаходить модуль, округлює число та обирає найменше й найбільше значення.

PYTHON · ПРИКЛАД

```python
distance = abs(12 - 20)
print(distance)
print(round(3.14159, 2))
print(min(5, 2, 9))
print(max(5, 2, 9))
print(pow(2, 3))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
8
3.14
2
9
8
```

Пояснення

abs дає модуль різниці, min і max обирають крайні значення, pow(2, 3) обчислює 2^3. round округлює число, але не задає кількість знаків у тексті. Для виведення рівно двох знаків використовують формат .2f.

Точність і особливості чисел

Приклад показує похибку обчислень із float і особливості округлення.

PYTHON · ПРИКЛАД

```python
value = 0.1 + 0.2
print(value)
print(round(value, 2))
print(round(2.5), round(3.5))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
0.30000000000000004
0.3
2 4
```

Пояснення

Деякі десяткові дроби неточно подаються у float, тому сума містить малу похибку. round створює округлений результат. На середині між цілими 2.5 і 3.5 округлюються до парних 2 і 4.

Введення та виведення

Отримаємо дані від користувача і оформимо результат.

Обчислення стає корисним, коли має вхід і вихід.

print(): призначення функції

print() — вбудована функція для виведення значень у текстовий потік, зазвичай у консоль.

Функція отримує значення, перетворює їх на текст і виводить. Можна передати одне значення, кілька значень або жодного.

Що виводимо

Текст, числа та результати виразів. Значення передаємо в дужках.

Як оформлюємо

sep розділяє значення, end завершує виведення.

Що повертається

Результат виклику — None. Виведення не створює рядок для обчислень.

print() без значень виводить лише завершення рядка.

print(): синтаксис виклику

Значення передають за позицією, а параметри оформлення — за іменем. sep і end необов’язкові.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
print(значення_1, значення_2,
      sep=роздільник, end=завершення)
```

Значення

Значення для виведення. Розділяються комами.

Роздільник (sep)

роздільник (sep) — текст між значеннями. Типово один пробіл.

Завершення (end)

завершення (end) — текст після значень. Типово перенос рядка.

Наприклад, sep=" | " змінює роздільник, а end=" " продовжує виведення в тому самому рядку.

print(): параметри виведення

Параметр	Типова поведінка	Коли змінюємо
`*objects`	Довільна кількість значень	print(name, score)
`sep=" "`	Пробіл між значеннями	sep=" | " для розділення
`end="\n"`	Новий рядок після виклику	end=" " для продовження рядка
`file=None`	Поточний sys.stdout	Виведення у відкритий файл
`flush=False`	Не примушує скидання буфера	flush=True для негайного виведення

sep, end, file, flush передають за іменем; print() повертає None.

Функція print()

Програма виводить кілька значень і змінює роздільник та завершення виведення.

PYTHON · ПРИКЛАД

```python
name = "Олена"
score = 85
print(name, score)
print(name, score, sep=" | ")
print("Результат:", end=" ")
print(score)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Олена 85
Олена | 85
Результат: 85
```

Пояснення

Перший print розділяє ім’я та оцінку пробілом. sep змінює роздільник на « | ». end=" " прибирає перенос, тому наступний print дописує 85 у той самий рядок.

input(): призначення та синтаксис

input() показує запрошення, очікує введення з клавіатури й повертає введений текст як str.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
текст = input(запрошення)

число = int(input(запрошення))
```

Запрошення

Необов’язковий текст перед введенням, наприклад "Вік: ".

Результат

Введені цифри теж є текстом. Enter у результат не входить.

Числове введення

int — ціле число, float — дробове. Потрібний допустимий текст.

Для float вводять десяткову крапку, наприклад 12.5. Текст 12,5 спричинить ValueError.

Функція input()

Програма зчитує вік користувача й обчислює його вік наступного року.

PYTHON · ПРИКЛАД

```python
text = input("Вік: ")
age = int(text)
print("Наступного року:", age + 1)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Вік: 18
Наступного року: 19
```

Пояснення

Після введення 18 функція input повертає текст «18». int створює число, до якого можна додати 1. У консолі 18 вводить користувач, а повідомлення про наступний рік виводить програма.

f-рядок: синтаксис підстановки

f-рядок створює текст із результатами обчислення виразів. Префікс f ставлять перед лапками.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
f"Текст {вираз}"

f"Текст {вираз:формат}"
```

f перед лапками

Вмикає обчислення виразів усередині фігурних дужок.

{вираз}

Місце підстановки: ім’я або вираз для обчислення.

:формат

Необов’язковий формат. .2f задає два знаки після крапки.

Формат змінює текстове подання результату, а не початкове число.

f-рядки: коротка довідка форматів

Запис	Призначення	Приклад результату
`f"{x}"`	Текстове подання виразу	x = 12 → "12"
`f"{x:.2f}"`	Два знаки після крапки	x = 3.5 → "3.50"
`f"{x:8.2f}"`	Ширина щонайменше 8 символів	Вирівнювання числа праворуч
`f"{x:04d}"`	Ціле число з нулями	x = 7 → "0007"
`f"{s:<10}"`	Текст ліворуч у полі ширини 10	> праворуч; ^ по центру
`f"{x:.1%}"`	Відсоткове подання	x = 0.25 → "25.0%"

Форматування створює рядок і не змінює початкове числове значення.

Форматування f-рядками

Програма підставляє значення у текст і форматує число з потрібною точністю.

PYTHON · ПРИКЛАД

```python
name = "Олена"
score = 83.456
print(f"{name}: {score:.2f}")
print(f"Після бонусу: {score + 2:.1f}")
print(score)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Олена: 83.46
Після бонусу: 85.5
83.456
```

Пояснення

У фігурних дужках обчислюється вираз. .2f показує оцінку з двома знаками, .1f — результат додавання з одним. Значення score залишається 83.456: форматування його не змінює.

Лінійна програма

Програма зчитує ціну та кількість товару, обчислює й виводить його вартість.

PYTHON · ПРИКЛАД

```
price = float(input("Ціна: "))
quantity = int(input("Кількість: "))
total = price * quantity
print(f"Вартість: {total:.2f} грн")
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Ціна: 12.5
Кількість: 3
Вартість: 37.50 грн
```

Пояснення

Введення ціни перетворюється на float, кількості — на int. Добуток 12.5 × 3 дорівнює 37.5. Формат .2f виводить 37.50 грн. Програма виконує послідовність введення, обчислення та виведення.

Рядки та обробка тексту

Розглянемо символи, зрізи та методи роботи з текстом.

input() повертає рядок — дослідимо цей тип докладніше.

Рядок str: поняття та властивості

Рядок str — незмінна впорядкована послідовність Unicode-символів. Він зберігає текстові дані.

Рядок може містити літери, цифри, пробіли та спеціальні символи. Текст числа не є числом: рядок «80» і число 80 мають різні типи.

Порядок

Кожен символ має позицію. Індекс першого символу — 0.

Незмінність

Окремий символ не можна замінити на місці. Перетворення дає новий рядок.

Застосування

Імена, повідомлення, вміст файлів і введення користувача.

Далі розберемо створення рядка, а потім — індекси, зрізи та методи.

Рядок: синтаксис створення

Текст записують у лапках. Одинарні та подвійні лапки рівноправні й мають бути парними.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
ім’я = "текст"
ім’я = 'текст'
ім’я = """багаторядковий текст"""
ім’я = str(значення)
```

Лапки

Межі тексту. Самі лапки не входять у рядок.

Потрійні лапки

Дозволяють переноси рядків безпосередньо в коді.

str(значення)

Текстове подання об’єкта. Порожній рядок — "".

Для читання символу та частини рядка використовують індекс і зріз.

Створення рядків

Приклад демонструє створення порожніх, звичайних і багаторядкових рядків.

PYTHON · ПРИКЛАД

```python
first = "Олена"
second = 'Python'
empty = ""
message = """Перший рядок
Другий рядок"""
print(first, second)
print(message)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Олена Python
Перший рядок
Другий рядок
```

Пояснення

Одинарні та подвійні лапки створюють однаковий тип str. Порожні лапки задають порожній рядок. Потрійні лапки зберігають перенос між двома рядками повідомлення.

Спеціальні символи

Приклад показує перенесення рядка, табуляцію та запис спеціальних символів.

PYTHON · ПРИКЛАД

```python
print("Перший\nДругий")
print("Ім\u0027я:\tОлена")
print("Вона сказала: \"Так\"")
path = r"C:\notes\data.txt"
print(path)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Перший
Другий
Ім'я:   Олена
Вона сказала: "Так"
C:\notes\data.txt
```

Пояснення

Послідовності \n і \t задають перенос та табуляцію. Екрановані лапки стають частиною тексту. Префікс r у шляху зберігає зворотні скісні риски як звичайні символи.

Індекс і зріз: синтаксис доступу

Індекс обирає один елемент послідовності. Зріз створює її частину за межами та кроком.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
послідовність[індекс]

послідовність[початок:кінець:крок]
```

Індекс

0 — перший, −1 — останній.
Поза межами — IndexError.

Початок і кінець

Початок входить, кінець — ні.
Межі можна пропустити.

Крок

Типово 1. Від'ємний — зворотний напрям, 0 заборонений.

Схематичні позначення замінюють конкретними іменами та числами.

Індексація рядків

Програма звертається до символів рядка за додатними й від'ємними індексами.

PYTHON · ПРИКЛАД

```python
word = "Python"
print(len(word))
print(word[0])
print(word[-1])
# word[6] → IndexError
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
6
P
n
```

Пояснення

У «Python» шість символів. Індекс 0 обирає P, а −1 — n з кінця. Найбільший додатний індекс — 5. Звернення word[6] спричинило б IndexError.

Зрізи

Програма створює частини рядка та змінює напрям і крок перебору символів.

PYTHON · ПРИКЛАД

```python
word = "Python"
print(word[:3])
print(word[-2:])
print(word[1:5:2])
print(word[::-1])
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Pyt
on
yh
nohtyP
```

Пояснення

[:3] бере позиції 0–2, а [-2:] — два останні символи. У [1:5:2] обираються позиції 1 і 3. Крок −1 у [::-1] читає весь рядок у зворотному напрямі.

Операції та незмінність

Приклад показує операції з рядками та обмеження, пов’язані з їхньою незмінністю.

PYTHON · ПРИКЛАД

```python
text = "кіт"
new_text = "л" + text[1:]
print(text, new_text)
print("ha" * 3)
print("іт" in text)
# text[0] = "л" → TypeError
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
кіт літ
hahaha
True
```

Пояснення

Зріз text[1:] дає «іт». Додавання «л» створює новий рядок «літ», а text залишається «кіт». * повторює текст, in перевіряє підрядок. Заміна символу на місці заборонена.

Функція й метод: синтаксис виклику

Функція

функція(об’єкт)

```
text = "  Python  "
print(len(text))
# 10
```

text передаємо аргументом у дужках. len() враховує також пробіли.

Метод

об’єкт.метод(аргументи)

```
clean = text.strip()
print(clean)
# Python
```

Метод викликаємо через об’єкт і крапку. strip() повертає новий рядок без пробілів по краях.

Початковий text не змінюється. Результат strip() зберігаємо у clean.

Рядки: очищення та пошук

Метод	Результат	Параметри / особливості
`strip(chars=None)`	Новий рядок без крайових символів	chars — набір символів, не підрядок
`lower(); upper()`	Новий рядок в іншому регістрі	Початковий рядок не змінюється
`find(sub, start, end)`	Позиція або -1	start/end можна не задавати
`index(sub, start, end)`	Позиція підрядка	start/end необов'язкові; немає збігу → ValueError
`count(sub, start, end)`	Кількість неперекривних збігів	start/end необов'язкові; задають ділянку пошуку
`startswith(prefix)`	True або False	endswith(suffix) перевіряє кінець

find() повертає 0 для збігу на початку — це теж успішний пошук.

Нормалізація та пошук

Приклад показує очищення тексту, зміну регістру, пошук і перевірку його країв.

PYTHON · ПРИКЛАД

```python
text = "  Python Course  "
clean = text.strip().lower()
print(clean)
print(clean.find("course"))
print(clean.count("o"))
print(clean.startswith("python"))
print("report.csv".endswith(".csv"))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
python course
7
2
True
True
```

Пояснення

strip прибирає пробіли по краях, lower створює текст у нижньому регістрі. find повертає індекс початку «course» — 7. count рахує два входження «о», а перевірки країв дають bool.

Рядки: поділ, заміна й об’єднання

Метод	Параметри	Важливий нюанс
`split(sep=None, maxsplit=-1)`	Роздільник; максимум поділів	Без sep — за пробільними символами
`rsplit(sep=None, maxsplit=-1)`	Поділ із правого краю	Особливо корисний із maxsplit
`replace(old, new, count)`	Що замінити; на що; кількість	Без count — усі входження
`sep.join(iterable)`	Послідовність рядків	Усі елементи мають бути str
`splitlines()`	Поділ на рядки	keepends=True зберігає переноси

"a,,b".split(",") зберігає порожній елемент; split() має інші правила.

Поділ і об’єднання

Програма розбиває рядок на частини й об’єднує їх із потрібним роздільником.

PYTHON · ПРИКЛАД

```
text = "80,75,90"
parts = text.split(",")
print(parts)
print(" | ".join(parts))
print(text.replace(",", ";"))
print(" a  b ".split())
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
['80', '75', '90']
80 | 75 | 90
80;75;90
['a', 'b']
```

Пояснення

split(",") створює список трьох рядків. join поєднує їх новим роздільником. replace повертає текст із заміненими комами. split() без аргументів інакше обробляє пробіли.

Списки й кортежі

Зберігатимемо кілька значень в одній структурі.

Від послідовності символів — до послідовності значень.

Список list: поняття та властивості

Список list — змінна впорядкована послідовність об’єктів. Він зберігає кілька значень в одній колекції.

Елементи списку можуть повторюватися й мати різні типи. Порядок важливий: перший елемент доступний за індексом 0.

Доступ

Індекс обирає елемент, зріз — частину списку.

Зміни

Можна замінювати, додавати та видаляти елементи.

Застосування

Оцінки, результати обчислень, список записів.

Список зручний, коли набір даних потрібно змінювати під час роботи.

Список: синтаксис створення та доступу

Квадратні дужки створюють список. Елементи відділяють комами, доступні позиції починаються з 0.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
список = [елемент_1, елемент_2]
список = list(джерело)
значення = список[індекс]
список[індекс] = нове_значення
```

[елементи]

Елементи розділяються комами. [] — порожній список.

list(джерело)

Збирає елементи джерела в новий список.

[індекс]

Читання або заміна елемента. Зріз — [початок: кінець].

Заміна за індексом змінює список. Додавання й видалення виконуємо відповідними методами.

Список і його елементи

Програма читає елементи списку та визначає його довжину.

PYTHON · ПРИКЛАД

```python
scores = [80, 75, 90]
print(scores[0])
print(scores[-1])
print(len(scores))
print(75 in scores)
empty = []
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
80
90
3
True
```

Пояснення

Перший та останній елементи доступні за індексами 0 і −1. len повертає кількість елементів — 3. in перевіряє наявність значення 75. [] створює окремий порожній список.

Зміна та зрізи списку

Приклад показує заміну елементів списку та роботу з його зрізами.

PYTHON · ПРИКЛАД

```python
scores = [80, 75, 90]
scores[1] = 85
part = scores[:2]
part[0] = 0
print(scores)
print(part)
scores[1:3] = [95]
print(scores)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
[80, 85, 90]
[0, 85]
[80, 95]
```

Пояснення

Заміна scores[1] змінює список. Зріз створює окремий список part, тому його зміна не змінює scores із чисел. Присвоєння зрізу замінює два елементи одним і скорочує список.

append() і extend(): що додається

append()

Один об’єкт у кінець

```
a = [1, 2]
a.append([3, 4])
print(a)
# [1, 2, [3, 4]]
```

Список [3, 4] додається цілком. Третім елементом a стає вкладений список.

extend()

Елементи по одному

```
b = [1, 2]
b.extend([3, 4])
print(b)
# [1, 2, 3, 4]
```

Метод перебирає аргумент і додає кожен елемент. У b отримуємо чотири числа.

Обидва методи змінюють список і повертають None. Для перевірки друкуємо сам список.

Списки: додавання й видалення

Метод / запис	Дія	Повернення / нюанс
`append(x)`	Додати один об’єкт у кінець	Повертає None
`extend(iterable)`	Додати всі елементи	Повертає None
`insert(i, x)`	Додати перед позицією i	Повертає None
`remove(x)`	Видалити перший збіг x	ValueError, якщо x немає
`pop(i=-1)`	Видалити елемент за індексом	Повертає елемент; типово останній
`clear(); del a[i]`	Очистити / видалити позицію	Не повертають видалений елемент

append([3, 4]) додає вкладений список; extend([3, 4]) додає два числа.

Додавання та видалення

PYTHON · ПРИКЛАД

```python
a = [1, 2]
a.append([3, 4])
b = [1, 2]
b.extend([3, 4])
b.insert(0, 0)
b.remove(2)
last = b.pop()
print(a)
print(b, last)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
[1, 2, [3, 4]]
[0, 1, 3] 4
```

Пояснення

append додає [3, 4] як один вкладений об’єкт, extend — як два числа. insert додає 0 на початок. remove видаляє значення 2, а pop забирає останній елемент і повертає 4.

Списки: пошук, копіювання, порядок

Метод	Призначення	Параметри / нюанс
`index(x, start, stop)`	Позиція першого збігу	start/stop необов'язкові; немає збігу → ValueError
`count(x)`	Кількість збігів	0, якщо значення немає
`copy()`	Поверхнева копія	Вкладені об'єкти лишаються спільними
`reverse()`	Змінити порядок на зворотний	Змінює список; повертає None
`sort(key=None, reverse=False)`	Сортувати на місці	key — функція; пояснення в лекції 2
`sorted(iterable, ... )`	Створити відсортований список	Це функція; початкові дані не змінює

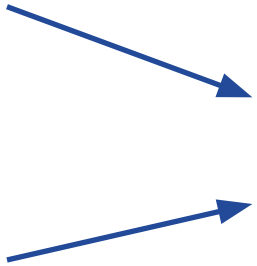
Не записуйте a = a.sort(): результат такого присвоєння — None.

Посилання та поверхнева копія

Два імені можуть посилатися на один список.

```
a = [80, 90]
```

```
b = a
```



Спільний список
[80, 90]

```
c = a.copy()
```
Окремий список

b.append(100) змінює список, який видно через a і b.

Поверхнева копія не копіює вкладені об’єкти рекурсивно.

Копії та вкладені структури

Приклад порівнює спільне посилання, копію списку та повторення вкладених списків.

PYTHON · ПРИКЛАД

```python
a = [80, 90]
b = a
c = a.copy()
b.append(100)
print(a, c)
matrix = [[0] * 2] * 2
matrix[0][0] = 1
print(matrix)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
[80, 90, 100] [80, 90]
[[1, 0], [1, 0]]
```

Пояснення

b і a посилаються на один список, c — на його копію. У matrix повторення створює два посилання на той самий рядок. Тому зміна одного елемента видима в обох рядках.

Кортеж tuple: поняття та властивості

Кортеж tuple — незмінна впорядкована послідовність об’єктів. Його склад не змінюється після створення.

Елементи можна читати за індексом і розпаковувати в окремі імена. Для іншого складу даних створюють новий кортеж.

Доступ

Індекси починаються з 0. Порядок і повтори зберігаються.

Незмінність

Елемент кортежу не можна замінити присвоєнням.

Застосування

Координати, фіксовані пари й групи пов’язаних значень.

Незмінність кортежу не забороняє змінювати вкладений список, якщо він є.

Кортеж: синтаксис створення та доступу

Кортеж утворюють значення, розділені комами. Дужки допомагають позначити межі запису.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
кортеж = (елемент_1, елемент_2)
один_елемент = (значення,)
порожній = ()
значення = кортеж[індекс]
ім'я_1, ім'я_2 = кортеж
перше, *решта = послідовність
```

Кома

(значення,) — кортеж. Без коми дужки лише групують вираз.

Індекс

Повертає елемент. Заміна за індексом заборонена.

Розпакування

Імена отримують елементи по порядку. *решта збирає залишок у список.

Для перетворення іншої послідовності на кортеж використовують tuple(джерело).

Кортежі

Приклад демонструє розпакування кортежу й роль коми в його створенні.

PYTHON · ПРИКЛАД

```python
point = (10, 20)
x, y = point
single = (5,)
print(x, y)
print(type(single).__name__)
print(type((5)).__name__)
# point[0] = 15 → TypeError
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
10 20
tuple
int
```

Пояснення

Розпакування дає x = 10 і y = 20. Кома в (5,) створює tuple, а (5) залишається int. type(...).__name__ повертає назву типу. Елемент point[0] можна прочитати, але не замінити присвоєнням.

Розпакування послідовностей

Програма розподіляє елементи послідовності між іменами за допомогою розпакування.

PYTHON · ПРИКЛАД

```python
scores = [80, 75, 90, 95]
first, *rest = scores
print(first)
print(rest)
start, *middle, end = scores
print(start, middle, end)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
80
[75, 90, 95]
80 [75, 90] 95
```

Пояснення

Зірочка збирає залишок значень у список. first отримує 80, rest — решту. У другому записі start і end беруть крайні елементи, а middle — два значення між ними.

Кортежі: операції та два методи

Запис	Призначення	Важливе правило
`t[i]; t[start:stop]`	Індекс; зріз	Доступ не змінює кортеж
`len(t); x in t`	Довжина; належність	Це функція та оператор
`t.count(x)`	Кількість входжень x	Повертає ціле число
`t.index(x, start, stop)`	Позиція першого збігу	start/stop необов’язкові; немає збігу → ValueError
`a, b = t`	Розпакування	Кількість значень має збігатися
`t + other; t * n`	Об’єднання; повторення	Створюють новий кортеж

У tuple немає append(), remove() або sort(): змінювати його структуру не можна.

Множини та словники

Працюватимемо з унікальними значеннями й записами.

Окрім позиції, дані можна знаходити за ключем.

Множина set: поняття та властивості

Множина set — змінна колекція унікальних хешованих елементів. Вона не має позиційних індексів.

Повторне додавання рівного елемента не збільшує множину. Для пошуку використовують перевірку належності, а не позицію.

Унікальність

Рівні значення представлені одним елементом. Порядок не гарантований.

Елементи

Числа й рядки підходять. Змінний список елементом set бути не може.

Застосування

Унікальні слова, спільні значення та різниця наборів.

Хешованість — придатність об’єкта для пошуку за хешем; рівність визначає повтори.

Множина: синтаксис створення та доступу

Фігурні дужки без пар «ключ — значення» створюють непорожню множину. Позиційного доступу немає.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
множина = {елемент_1, елемент_2}
множина = set(джерело)
порожня = set()
елемент in множина
```

{елементи}

Рівні елементи не зберігаються як окремі повтори.

set(джерело)

Створює множину з джерела. set() — порожня множина.

in

Належність: True або False. Індексів у множини немає.

Важлива відмінність: {} створює порожній словник, а не множину.

Множини

Приклад показує усунення повторів і створення порожньої множини.

PYTHON · ПРИКЛАД

```python
values = [3, 1, 3, 2]
unique = set(values)
print(len(unique))
print(3 in unique)
empty = set()
print(type(empty).__name__)
print(type({}).__name__)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
3
True
set
dict
```

Пояснення

set прибирає повтори, тому unique містить три числа. in перевіряє наявність 3. set() створює порожню множину, а {} — словник. type(...).__name__ повертає назву типу.

Множини: методи та операції

Метод / оператор	Дія	Поведінка
`add(x); update(iterable)`	Один / багато елементів	Змінює множину; None
`remove(x); discard(x)`	Видалити x	Немає елемента x → remove породжує KeyError
`pop(); clear()`	Видалити довільний / усі	pop не означає «останній»
`a.union(b); a | b`	Об’єднання	Нова множина
`a.intersection(b); a & b`	Перетин	Нова множина
`a.difference(b); a - b`	Різниця	Нова множина
`a.symmetric_difference(b)`	Лише неспільні елементи	Також оператор a ^ b

a, b — множини; x — елемент; iterable — джерело для перебору. Порядок друку set не гарантований.

Операції множин

Приклад демонструє перетин, різницю, об’єднання та зміну множин.

PYTHON · ПРИКЛАД

```python
a = {1, 2, 3}
b = {3, 4}
print(a & b)
print(a - b)
print(len(a | b))
a.add(5)
a.discard(9)
print(5 in a)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
{3}
{1, 2}
4
True
```

Пояснення

Перетин залишає спільне 3, різниця — лише елементи a поза b. Об’єднання має чотири елементи. add додає 5, discard(9) не спричиняє помилки, коли 9 немає.

Словник dict: поняття та властивості

Словник dict — змінна колекція пар «ключ — значення». Значення знаходять за ключем.

Ключі унікальні. Присвоєння за наявним ключем оновлює значення, а за новим — додає пару. Порядок вставлення зберігається.

Ключ

Називає дані. Має бути хешованим, наприклад числом або рядком.

Значення

Може бути числом, рядком, списком чи іншим об’єктом.

Застосування

Запис студента, налаштування, підрахунок за назвою.

Список використовує позиції, словник — ключі. Ключ не обов’язково є рядком.

Словник: синтаксис створення та доступу

Фігурні дужки містять пари. Двокрапка відділяє ключ від значення, коми розділяють пари.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
словник = {ключ_1: значення_1,
           ключ_2: значення_2}
значення = словник[ключ]
словник[ключ] = нове_значення
порожній = {}
```

Ключ: значення

Двокрапка розділяє ключ і значення. Коми розділяють пари.

Читання [ключ]

Повертає дані за ключем. Відсутній ключ дає KeyError.

Присвоєння

Оновлює наявну пару або додає нову.

Для читання із запасним значенням використовують словник.get(ключ, заміна).

Словник

Приклад показує читання й оновлення значень словника за ключами.

PYTHON · ПРИКЛАД

```python
student = {
    "name": "Олена",
    "score": 85,
}
print(student["name"])
student["score"] = 90
print(student["score"])
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Олена
90
```

Пояснення

Ключ «name» повертає ім’я Олена. Присвоєння за ключем «score» оновлює оцінку 85 до 90. Інша пара залишається без змін. Квадратні дужки тут містять ключ, а не позицію.

Словники: доступ і робота з полями

Метод	Результат	Параметри / нюанс
`get(key, default=None)`	Значення або заміна	Відсутній ключ не додає
`setdefault(key, default=None)`	Значення наявного / нового ключа	Якщо ключа немає — додає його
`update(other, **kwargs)`	Оновити пари ключ → значення	Змінює словник; повертає None
`pop(key, default)`	Видалити й повернути значення	default необов'язковий; немає ключа й default → KeyError
`keys(); values(); items()`	Подання ключів, значень, пар	Це не готові незалежні списки
`copy(); clear()`	Поверхнева копія / очищення	copy повертає dict; clear — None
`popitem()`	Видалити останню вставлену пару	Порожній словник → KeyError

in перевіряє ключі; для пошуку серед значень використовуйте values().

Оновлення й безпечний доступ

Приклад демонструє безпечний доступ, додавання та видалення полів словника.

PYTHON · ПРИКЛАД

```python
student = {"name": "Олена"}
print(student.get("score", 0))
student.update({"score": 85})
print("score" in student)
score = student.pop("score")
print(score)
print("score" in student)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
0
True
85
False
```

Пояснення

get повертає запасний 0, бо поля score ще немає. update додає оцінку. pop видаляє пару й повертає 85. Перевірка in шукає ключ, тому після видалення дає False.

Подання та вкладені записи

Приклад показує подання словника та доступ до полів вкладених записів.

PYTHON · ПРИКЛАД

```python
student = {"name": "Олена", "score": 85}
print(list(student.keys()))
print(list(student.values()))
print(list(student.items()))
records = [
    student, {"name": "Ігор", "score": 70}
]
print(records[1]["name"])
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
['name', 'score']
['Олена', 85]
[('name', 'Олена'), ('score', 85)]
Ігор
```

Пояснення

keys, values та items дають подання ключів, значень і пар. list збирає їх для показу. records[1] обирає другий словник, а ["name"] читає його поле — Ігор.

Як обрати колекцію

Структуру даних обирають за способом доступу та допустимими змінами.

list / tuple

Послідовність і доступ за індексом. list змінний; tuple незмінний.

set

Унікальні елементи, перевірка належності; немає індексів.

dict

Пари ключ → значення; доступ за ключем; порядок вставлення.

Оцінки → list; координати → tuple; унікальні слова → set; запис → dict.

Колекції: коротке порівняння

Тип	Доступ і порядок	Зміни / повтори
`str`	Індекс; порядок символів	Незмінний; повтори можливі
`list`	Індекс; порядок елементів	Змінний; повтори можливі
`tuple`	Індекс; порядок елементів	Незмінний; повтори можливі
`set`	Належність; немає індексів	Змінний; елементи унікальні
`dict`	Ключ; порядок вставлення	Змінний; ключі унікальні

«Незмінний» описує саму структуру; вкладений змінний об’єкт — окремий випадок.

Умови та логічні вирази

Навчимо програму обирати дію залежно від даних.

Тип даних обрано — тепер перевіряємо його значення.

Логічний вираз: умова вибору

Логічний вираз задає перевірку. Для вибору гілки Python оцінює істинність його результату.

Порівняння чисел дає True або False. Кілька перевірок поєднують операторами and, or та not. Дужки роблять порядок дій явним.

Порівняння

a == b — рівність. a != b — нерівність. <, <=, >, >= — порядок.

Поєднання

Для bool: and — обидві істинні. or — хоча б одна. not — заперечення.

Істинність

0, None та порожні колекції хибні. Непорожній рядок «0» істинний.

Для логічних виразів пріоритет такий: not, потім and, потім or.

Умова: синтаксис і вибір гілки

У ланцюжку if/elif/else виконується перша відповідна гілка.

if — якщо; elif — інакше, якщо; else — інакше.

```
if умова:
    інструкції_1
elif інша_умова:
    інструкції_2
else:
    інструкції_3
```



elif та else необов’язкові. Схема праворуч показує приклад із двома гілками.

Порівняння й логіка

Програма перевіряє діапазон оцінки та поєднує умови логічними операторами.

PYTHON · ПРИКЛАД

```python
score = 75
print(0 <= score <= 100)
print(score >= 60 and score <= 100)
print(score < 0 or score > 100)
print(not (score >= 60))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
True
True
False
False
```

Пояснення

Оцінка 75 належить діапазону 0–100. Обидві умови з and істинні. Жодна умова виходу за межі з or не виконується. not заперечує істинність score >= 60.

Умовна конструкція

PYTHON · ПРИКЛАД

```python
score = 75
if score >= 90:
    category = "Відмінно"
elif score >= 60:
    category = "Зараховано"
else:
    category = "Не зараховано"
print(category)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Зараховано
```

Пояснення

75 не досягає 90, але задовольняє другу умову score >= 60. Тому обирається гілка elif. Інші гілки цього ланцюжка не виконуються, результат — «Зараховано».

Умовний вираз і коротке обчислення

Умовний вираз обирає одне значення. and та or обчислюють наступний операнд лише за потреби.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
значення_так if умова else значення_ні

перший and другий
перший or другий
```

if … else

Істинна умова обирає значення_так, хибна — значення_ні.

and

Якщо перший операнд хибний — повертає його, інакше — другий.

or

Якщо перший операнд істинний — повертає його, інакше — другий.

and та or повертають операнд, тому їхній результат не обов’язково має тип bool.

Істинність та коротке обчислення

Приклад демонструє перевірку порожніх даних і вибір запасного значення через or.

PYTHON · ПРИКЛАД

```python
scores = []
if scores:
    print(sum(scores) / len(scores))
else:
    print("Немає оцінок")
name = ""
print(name or "Без імені")
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Немає оцінок
Без імені
```

Пояснення

Порожній scores є хибним, тому середнє не обчислюється й ділення на нуль не відбувається. Порожній name у виразі з or замінюється запасним текстом. or повертає операнд.

Вкладені умови й умовний вираз

PYTHON · ПРИКЛАД

```python
score = 75
if 0 <= score <= 100:
    if score >= 60:
        print("Успішний результат")
    else:
        print("Потрібне повторення")
else:
    print("Некоректна оцінка")
label = "Так" if score >= 60 else "Ні"
print(label)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Успішний результат
Так
```

Пояснення

Пояснення послідовності дій та умов — на наступному слайді.

Вкладені умови: розбір прикладу

У попередньому прикладі дві вкладені перевірки мають різні завдання. Спочатку визначають допустимість оцінки, потім — успішність.

Для score = 75 зовнішня та внутрішня умови істинні. Програма друкує «Успішний результат». Окремий умовний вираз надає label значення «Так».

Допустима оцінка

0 <= score <= 100 перевіряє межі. Його else обробляє некоректну оцінку.

Успішність

score >= 60 перевіряють усередині допустимого діапазону.

Умовний вираз

label обирається лише за порогом 60. Перевірки меж тут немає.

Умовний вираз не замінює перевірку допустимості. Наприклад, для 120 label матиме значення «Так».

== та is: значення й тотожність

==

Порівняння значень

```
a = [1, 2]
b = [1, 2]
print(a == b)
# True
```

Списки створено окремо, але вони містять однакові значення в однаковому порядку.

is

Тотожність об’єкта

```
c = a
print(a is b)  # False
print(a is c)  # True
```

c та a посилаються на один список. b посилається на інший об’єкт.

Для рівності чисел і тексту використовуємо ==. Для відсутності значення — is None.

Рівність, тотожність і належність

Приклад порівнює рівність значень, тотожність об’єктів і належність елемента.

PYTHON · ПРИКЛАД

```python
a = [1, 2]
b = [1, 2]
c = a
print(a == b)
print(a is b, a is c)
value = None
print(value is None)
print(2 in a, 3 not in a)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
True
False True
True
True True
```

Пояснення

a та b мають рівні значення, але є різними списками. c посилається на той самий об’єкт, що й a. is перевіряє тотожність, == — рівність. Для None використовують is None.

Цикли та перебір даних

Повторюватимемо дії й оброблятимемо набори записів.

Умова визначає дію, цикл повторює її для багатьох значень.

while і for: коли який цикл потрібний

Цикл повторює блок інструкцій. while керується умовою, for отримує елементи джерела.

while

```
while умова:
    інструкції
```

Умову перевіряють перед кожною ітерацією. Поки вона істинна, тіло повторюється. Потрібно забезпечити завершення.

for

```
for елемент in джерело:
    інструкції
```

Наступний елемент присвоюють імені. Тіло виконується для кожного елемента. Після вичерпання джерела цикл завершується.

Відступи визначають тіло; break завершує цикл, continue пропускає решту ітерації.

Цикл while

Програма повторює інструкції, доки умова циклу while залишається істинною.

PYTHON · ПРИКЛАД

```python
counter = 1
while counter <= 3:
    print(counter)
    counter += 1
print("Готово")
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
1
2
3
Готово
```

Пояснення

Лічильник набуває значень 1, 2 і 3. Після кожного друку його збільшують. Коли counter стає 4, умова хибна й цикл завершується. Останній print виконується поза циклом.

Цикл for

Програма перебирає оцінки й символи рядка за допомогою циклу for.

PYTHON · ПРИКЛАД

```python
scores = [80, 75, 90]
for score in scores:
    print(score)
for letter in "кіт":
    print(letter)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
80
75
90
к
і
т
```

Пояснення

Перший for послідовно отримує три оцінки. Другий перебирає символи рядка «кіт». Ім’я в заголовку щоразу отримує наступний елемент, тому ручний індекс не потрібний.

range(): призначення та синтаксис

range описує послідовність цілих чисел для перебору. Він не створює готовий список усіх значень.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
range(кінець)
range(початок, кінець)
range(початок, кінець, крок)
```

Початок (start)

Перше значення непорожнього діапазону. Типово — 0.

Кінець (stop)

Не входить у діапазон. Для чисел 1–5 потрібна межа 6.

Крок (step)

Типово 1. Може бути від’ємним, але не 0.

Аргументи мають бути цілими. list(range(...)) збирає значення в список для показу.

range(): параметри та межі

Форма	Параметри	Приклад значень
`range(stop)`	start=0; step=1	range(4) → 0, 1, 2, 3
`range(start, stop)`	step=1	range(2, 5) → 2, 3, 4
`range(start, stop, step)`	Довільний ненульовий крок	range(1, 6, 2) → 1, 3, 5
`range(5, 0, -1)`	Від'ємний крок	5, 4, 3, 2, 1
`range(5, 1)`	Початок більший за межу	Порожній діапазон за кроку +1
`range(0, 5, 0)`	Нульовий крок	ValueError

stop не входить до діапазону; range — опис послідовності, а не готовий список.

Діапазон range()

Приклад показує, як початок, кінець і крок визначають значення range().

PYTHON · ПРИКЛАД

```python
print(list(range(1, 6)))
print(list(range(5, 0, -1)))
for number in range(0, 6, 2):
    print(number)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
0
2
4
```

Пояснення

range не включає кінцеву межу. Межа 6 дає числа до 5, крок −1 — зворотний порядок. У циклі крок 2 обирає 0, 2 і 4. list збирає діапазон для показу.

Накопичення й пошук

PYTHON · ПРИКЛАД

```python
scores = [80, 45, 90]
total = 0
passed_count = 0
for score in scores:
    total += score
    if score >= 60:
        passed_count += 1
print(total, passed_count)
print(sum(scores), min(scores), max(scores))
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
215 2
215 45 90
```

Пояснення

total додає кожну оцінку й отримує 215. Лічильник збільшується лише для 80 та 90, тому успішних двоє. Вбудовані функції окремо дають суму, мінімум і максимум.

Керування циклом

PYTHON · ПРИКЛАД

```python
for number in [1, -2, 3, 4]:
    if number < 0:
        continue
    if number == 4:
        break
    print(number)
if True:
    pass
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
1
3
```

Пояснення

Для −2 continue пропускає друк. Для 4 break припиняє цикл, тому друкуються лише 1 і 3. pass є порожньою інструкцією й не завершує ні цикл, ні програму.

Перебір: номер, елемент і пари

enumerate додає номер до елемента. zip поєднує відповідні елементи кількох джерел. items() дає пари словника.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
enumerate(джерело, start=0)
zip(джерело_1, джерело_2)
словник.items()

for номер, елемент in enumerate(джерело):
    інструкції
```

enumerate

Пара (номер, елемент). start — початковий номер, типово 0.

zip

Пари з відповідних позицій. Межа — найкоротше джерело.

items()

Пари (ключ, значення). У for розпаковуються у два імені.

Ці засоби дають значення під час перебору. Нумерація enumerate не змінює індекси колекції.

Перебір записів

Приклад показує перебір записів за допомогою enumerate, items і zip.

PYTHON · ПРИКЛАД

```python
records = [{"name": "Олена", "score": 85}]
for index, record in enumerate(records, 1):
    print(index, record["name"])
for key, value in records[0].items():
    print(key, value)
for name, score in zip(["Ігор"], [70]):
    print(name, score)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
1 Олена
name Олена
score 85
Ігор 70
```

Пояснення

enumerate нумерує запис від 1. items розпаковує пари ключ–значення. zip поєднує ім’я з відповідною оцінкою. Номер, ключ і позиція — різні способи доступу до даних.

Керування циклом та його else

break завершує найближчий цикл. continue пропускає решту поточної ітерації. pass не виконує жодної дії.

СХЕМА · ПОЗНАЧЕННЯ ЗАМІНЮЮТЬ ДАНИМИ

```
for елемент in джерело:
    if умова_пропуску:
        continue
    if умова_зупинки:
        break
    інструкції
else:
    дії_без_break
```

else належить циклу на тому самому рівні відступу. Це не гілка внутрішнього if.

continue

Переходить до наступної ітерації, пропускаючи решту тіла.

break

Виходить із найближчого циклу та пропускає його else.

else циклу

Працює при завершенні без break, навіть без ітерацій.

Вкладені цикли та else циклу

PYTHON · ПРИКЛАД

```python
matrix = [[1, 2], [3, 4]]
for row in matrix:
    for value in row:
        print(value, end=" ")
    print()
for score in [40, 55]:
    if score >= 60:
        print("Знайдено")
        break
else:
    print("Успішних оцінок немає")
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
1 2
3 4
Успішних оцінок немає
```

Пояснення

Пояснення послідовності дій та умов — на наступному слайді.

Вкладені цикли та else: розбір

Приклад містить два незалежні фрагменти. Перший друкує матрицю рядок за рядком. Другий шукає першу оцінку не меншу за 60.

У матриці надруковано рядки «1 2» і «3 4». Оцінки 40 та 55 не відповідають умові, тому пошук завершується без break і виконує else.

Рядок матриці

Зовнішній for обирає список row. Він задає поточний рядок.

Елемент рядка

Внутрішній for друкує числа через пробіл. print() після нього починає новий рядок.

Завершення пошуку

break спрацьовує за першого збігу. Якщо збігу немає, працює else циклу.

else вирівняно з for пошуку. break завершує лише найближчий цикл, а не всі вкладені цикли.

Завершення першої лекції

Поєднаємо дані, перевірки й цикли в одній програмі.

Наступна лекція: функції та надійність програм.

Наскрізний приклад: оцінки

PYTHON · ПРИКЛАД

```python
records = [
    {"name": "Олена", "score": 85},
    {"name": "Ігор", "score": 45},
]
total = 0
for record in records:
    score = record["score"]
    if 0 <= score <= 100:
        total += score
        status = "Так" if score >= 60 else "Ні"
        print(record["name"], status)
print("Сума:", total)
```

КОНСОЛЬ · РЕЗУЛЬТАТ

```
Олена Так
Ігор Ні
Сума: 130
```

Пояснення

Пояснення послідовності дій та умов — на наступному слайді.

Обробка записів: розбір прикладу

records — список словників. У кожному записі ключі name і score задають ім’я та оцінку. Цикл послідовно обробляє ці записи.

Оцінки 85 і 45 належать діапазону 0–100, тому обидві додаються до total. Для Олени виведено «Так», для Ігоря — «Ні». Підсумкова сума дорівнює 130.

Доступ до даних

record["score"] читає оцінку поточного запису, record["name"] — ім’я.

Перевірка й сума

Лише допустимі оцінки додаються до total. Некоректний запис пропускається.

Статус

Умовний вираз порівнює оцінку з 60 та обирає «Так» або «Ні».

total = 0 — перед циклом. Підсумковий print — після циклу, тому суму виведено один раз.

Типові помилки

Помилка

Неузгоджені типи та межі

```
"80" + 5        # TypeError
int("8.5")      # ValueError
list(range(1, 5))
# [1, 2, 3, 4]
```

Рядок не додається до числа. int() не читає дробовий текст. Межа stop не входить у range.

Виправлення

Перетворення та правильна межа

```
int("80") + 5   # 85
float("8.5")    # 8.5
list(range(1, 6))
# [1, 2, 3, 4, 5]
```

Для цілого й дробового тексту обираємо відповідне перетворення. Для чисел 1–5 задаємо stop = 6.

Перед запуском перевіряємо тип даних, допустимі значення та межі діапазону.

Підсумок і самоперевірка

Від даних до алгоритму: тип → операція → умова → повторення.

Питання 1

Що поверне "Python"[1:4]? Чому права межа не входить?

Питання 2

Яка колекція потрібна для унікальних слів? А для запису студента?

Питання 3

Чим відрізняються while і for? Що робить continue?

Відповіді: "yth"; set / dict; умова / перебір; перехід до наступної ітерації.

Практика до лекції 2

Самостійна робота: звіт за списком оцінок студентів.

Вхід

Три записи: Олена — 85, Ігор — 45, Марія — 110.

Завдання

Відкинути оцінки поза 0–100. Порахувати суму та кількість успішних.

Результат

Допустимих: 2. Сума: 130. Успішних: 1. Додатково — середнє 65.00.

Далі: виділимо обчислення у функції й навчимося обробляти помилки.