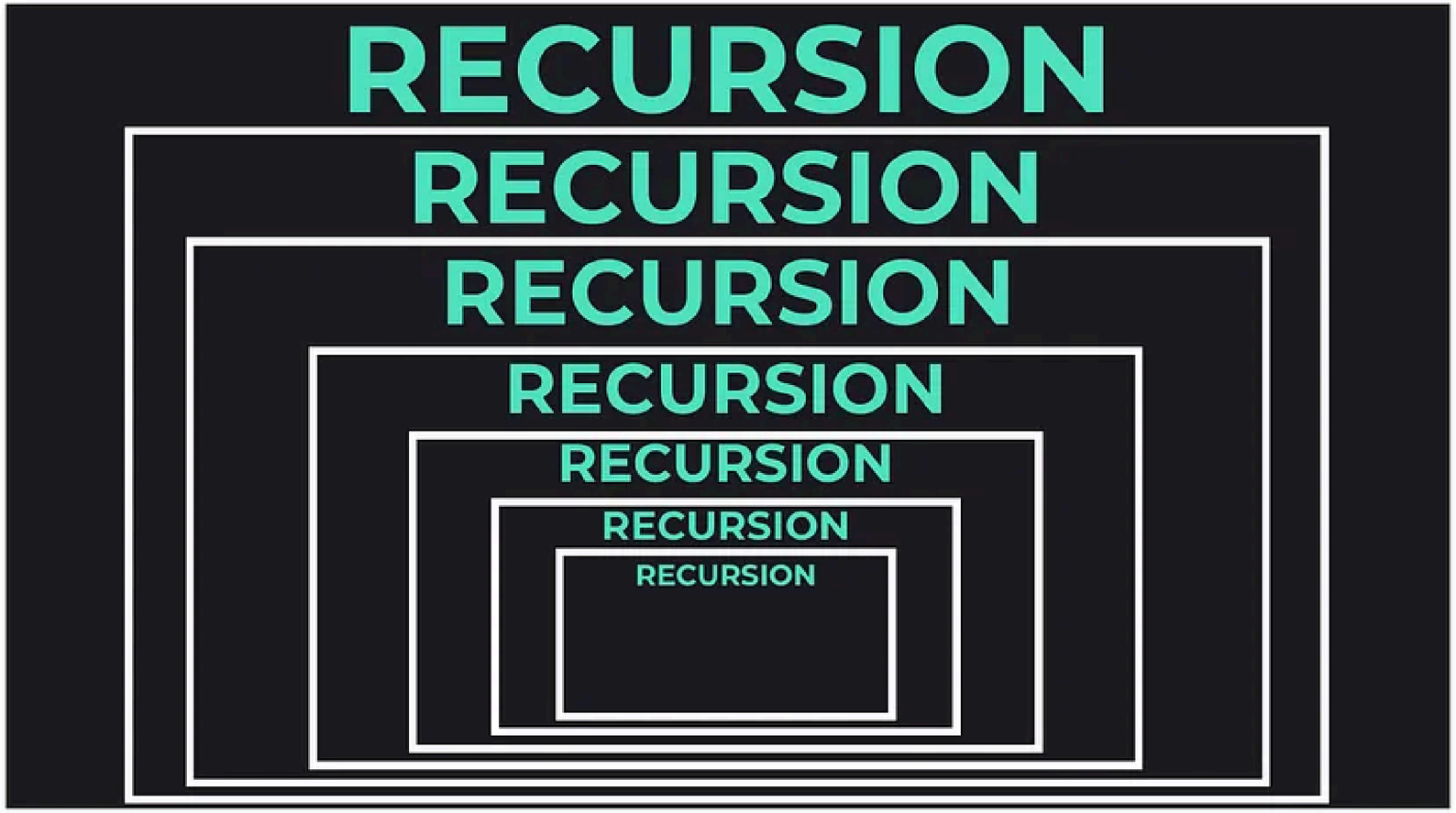
Умови та цикли

Рекурсія*



*див. Рекурсія

Конструкція if, elif, else

```
if умова:
    # Код виконується, якщо умова істинна
elif інша_умова:
    # Код виконується, якщо перша умова хибна, а ця істинна
else:
    # Код виконується, якщо всі попередні умови хибні
```

Умови порівняння

Оператор	Опис	Приклад
`==`	Рівність	`x == y`
`!=`	Нерівність	`x != y`
`<`	Менше	`x < y`
`>`	Більше	`x > y`
`<=`	Менше або дорівнює	`x <= y`
`>=`	Більше або дорівнює	`x >= y`

Порядок виконання логічних операцій

not — логічне заперечення, повертає булеве протилежне значення.
and — логічне **"І"** (всі умови мають бути істинними).
Повертає перше **False** чи останнє **True.**
or — логічне **"Або"** (хоча б одна умова має бути істинною)**.**
Повертає перше **True** чи останнє **False.**

Тип	Значення `False`	Значення `True`
int, float, complex	`0`, `0.0`, `0j`	Усі інші числа
bool	`False`	`True`
str	`''` (порожній рядок)	Усі непорожні рядки
list, tuple, set	Порожні колекції: `[]`, `()`, `set()`	Усі непорожні колекції
dict	`{}` (порожній словник)	Усі непорожні словники
None	`None`	Не застосовується
range	`range(0)`	Непорожні діапазони
object	`__bool__()` → `False` або `__len__()` → `0`	Усі інші об'єкти

Цикл for

Використовується для ітерації через елементи послідовностей, таких як списки, рядки, словники, діапазони тощо.

for variable (може бути будь-яка назва, окрім зарезервованих) **in послідовність**:

Цикл while

Використовується, коли потрібно виконувати блок коду, поки умова є істинною.

while condition:

break

Дозволяє достроково вийти з циклу.

```
for i in range(10):
        if i == 5:
            break
        print(i)
```

continue

У Python використовується для пропуску поточної ітерації циклу і переходу до наступної ітерації. Це означає, що весь код після continue в тілі циклу не виконується для поточної ітерації.

```
for number in range(10):
        if number % 2 == 0: # Якщо число парне
                continue
        print(f"Odd Number: {number}")
```

else

Блок **else** виконується після завершення циклу за умови, що цикл не був перерваний оператором **break**.
Якщо в циклі використовується **break**, блок **else** не виконується.
Оператор **continue**, навпаки, не впливає на виконання блоку **else**.

Загальні рекомендації

Вибирайте for для ітерації через послідовності та while, якщо умова є динамічною.

Використовуйте break та continue для кращого контролю над циклом.

Уникайте нескінченних циклів без чіткої умови виходу.

Рекомендований максимальний рівень вкладеності

PEP 8, офіційний стиль Python, не обмежує вкладеність, але хорошою практикою вважається не перевищувати 3-4 рівні.

Якщо ваш код виглядає надто складним, подумайте про рефакторинг: розбийте його на функції або використовуйте допоміжні структури даних.

Рекурсія

Рекурсія — це техніка програмування, коли функція викликає сама себе для розв'язання задачі. Вона часто використовується для обробки задач, які можуть бути розділені на менші підзадачі подібного типу.

У Python рекурсія реалізується через звичайний виклик функції, і кожен рекурсивний виклик створює новий контекст виконання функції.

Основні частини рекурсивної функції func(n)

Базовий випадок:

Умова, при якій рекурсія завершується.

Наприклад:

```
if n = 1:
    return 1
```

Без базового випадку функція викликатиме себе нескінченно (або до досягнення глибини стека).

Рекурсивний виклик:

Виклик функції всередині самої себе з простішими (меншими) параметрами.

```
def func(n):
    ....
    func(n-1)
    .....
```

Рекурсія vs Ітерація

Будь-яку рекурсію можна переписати через цикл (ітерацію).

Обмеження рекурсії в Python

Python має обмеження на максимальну глибину рекурсії. За замовчуванням це значення становить 1000 рівнів.

```
import sys
print(sys.getrecursionlimit()) # Результат: 1000

sys.setrecursionlimit(2000) # Змінити обмеження
```

Збільшення ліміту може призвести до перевантаження пам'яті.

Рекомендації для використання рекурсії

1. Завжди вказуйте базовий випадок, щоб уникнути нескінченної рекурсії.
2. Якщо задача може бути розв'язана ітеративно, обирайте ітерацію для кращої продуктивності.
3. Використовуйте мемоізацію для оптимізації рекурсії (збереження результатів попередніх викликів).