# Отступы

Отступы в Python используются для определения блоков кода. В отличие от многих других языков программирования, где блоки кода обозначаются скобками, в Python для этого используются отступы. Правильное использование отступов важно, поскольку они напрямую влияют на выполнение кода.
Если не соблюдать отступы, Python выдаст ошибку **IndentationError**.

**Как работают отступы**

- Блок кода под условием, циклом или определением функции должен начинаться с отступа. Обычно используются четыре пробела для каждого уровня отступа, хотя можно использовать и табуляцию. Главное — соблюдать единообразие во всем коде.

- Все инструкции внутри блока должны иметь одинаковый отступ. Это говорит Python, что они находятся на одном уровне вложенности и будут выполняться вместе.

- При вложенных конструкциях (например, условные операторы внутри циклов) используются дополнительные отступы для обозначения вложенности.

- Используйте либо пробелы, либо табуляцию для отступов, но не смешивайте их в одном файле. По умолчанию рекомендуется использовать 4 пробела на один уровень отступа.

- Соблюдение стиля отступов важно для поддержания читаемости кода и его правильного выполнения.

```
if 5 > 2:
    print("Пять больше, чем два!")  # Этот код выполнится, так как условие истинно
```

```python
for i in range(5):
    if i % 2 == 0:
        print(f"{i} - четное число")
    else:
        print(f"{i} - нечетное число")
```

```python
def say_hello(name):
    greeting = f"Привет, {name}!"
    print(greeting)

say_hello("Алиса")
```

## Скобки

Скобки в Python используются для различных целей, включая группировку выражений, определение приоритета операций, создание списков, кортежей, множеств и словарей, а также для вызова функций.

Примеры:

Группировка выражений: (1 + 2) * 3
Список: [1, 2, 3]
Кортеж: (1, 2, 3)
Множество: {1, 2, 3}
Словарь: {'ключ': 'значение'}
Вызов функции: print("Привет, мир!")

## Знаки пунктуации

**Точка с запятой (;)**: В Python точка с запятой используется редко. Она может разделять несколько инструкций, написанных в одной

строке, но обычно рекомендуется писать каждую инструкцию на новой строке.

```
x = 5; y = 10; print(x + y)
```

Однако лучше следовать рекомендациям по стилю и писать каждую инструкцию на новой строке, особенно в больших и сложных программах, где читаемость кода становится критически важной:

```
x = 5
y = 10
print(x + y)
```

**Двоеточие (:)**: Двоеточие используется в Python для обозначения начала блока кода, следующего за конструкциями управления потоком (if, for, while, def и т.д.).

```
if 5 > 2:
    print("Пять больше, чем два!")
```

**Кавычки (" или "")**: В Python для обозначения строк можно использовать как одинарные, так и двойные кавычки. Это позволяет включать один тип кавычек в строку, не используя экранирование.
'Это строка в одинарных кавычках.'
"Это строка в двойных кавычках."
"Он сказал: 'Привет!' "
'Она ответила: "Привет!" '

## Комментарии

Комментарии в Python начинаются с символа #. Всё, что следует за # в той же строке, Python игнорирует. Это позволяет добавлять описания к коду, что улучшает его читаемость.

```
# Это комментарий
print("Этот код выполнится.")
```

```
# print("Этот код не выполнится.")
```

## Списковые включения (List Comprehensions)

Python поддерживает компактный способ создания списков, который называется списковыми включениями.

```
squares = [x**2 for x in range(10)]
```

## Множественное присваивание

В Python можно одновременно присваивать значения нескольким переменным.

```
x, y, z = 1, 2, 3
```

## Распаковка коллекций

Python позволяет извлекать элементы из списков, кортежей или других итерируемых объектов в переменные.

```
coordinates = (1, 2, 3)
x, y, z = coordinates
```

## Использование _ для ненужных переменных

Когда вам нужно игнорировать определённые значения при распаковке, можно использовать _.

```
x, _, z = (1, 2, 3)  # 2 будет проигнорировано
```

## Анонимные (lambda) функции

Lambda-функции позволяют создавать анонимные функции на лету, они особенно удобны для использования в качестве аргументов там, где требуются функции.

```
double = lambda x: x * 2
print(double(5))
```

## Декораторы

Декораторы предоставляют простой способ модификации поведения функций.

```
def my_decorator(func):
    def wrapper():
        print("Что-то происходит перед вызовом функции.")
        func()
        print("Что-то происходит после вызова функции.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")

say_hello()
```

## Цикл while

Цикл while выполняется, пока условие истинно. Он используется, когда количество итераций неизвестно перед началом цикла.

Синтаксис:

```
while условие:
    # блок кода для выполнения
```

```
x = 0
while x < 5:
    print(f"x = {x}")
    x += 1  # Эквивалентно x = x + 1
```

## Цикл for

Цикл for в Python итерирует по элементам любой последовательности (например, списка или строки) в порядке, в котором они появляются в последовательности.

Синтаксис:

```
for переменная in последовательность:
    # блок кода для выполнения
```

```
for i in range(5):  # Функция range(5) генерирует последовательность от 0 до 4
    print(f"i = {i}")
```

## Условные конструкции

Условные конструкции (if, elif, else) используются для выполнения кода в зависимости от того, истинно или ложно выражение.

```
if условие1:
    # блок кода, если условие1 истинно
elif условие2:
    # блок кода, если условие2 истинно
else:
    # блок кода, если ни одно из условий выше не истинно
```

```
x = 10
if x > 10:
```

```
    print("x больше 10")
elif x < 10:
    print("x меньше 10")
else:
    print("x равен 10")
```