Циклы в Python + break / continue / pass

1. Что такое цикл?

В программировании очень часто нужно повторять одно и то же действие много раз: посчитать сумму чисел, несколько раз запросить данные у пользователя, пройтись по символам строки и т.д. Чтобы не копировать одну и ту же строку кода десятки раз, в языках программирования существует конструкция «цикл». Цикл — это блок кода, который выполняется многократно, пока выполняется определённое условие или пока мы сами его не остановим.

В Python есть два основных вида циклов:

• цикл for — когда мы заранее знаем, сколько раз нужно повторить действие, или хотим пройтись по каждому элементу последовательности (например, по символам строки);

• цикл while — когда мы повторяем действие «пока условие истинно» (например, пока пользователь не введёт правильный пароль).

2. Цикл for и функция range()

Цикл for перебирает последовательность значений. Чаще всего с числами в Python мы используем его вместе с функцией range(). Функция range() не создаёт список, а даёт последовательность чисел, по которой можно пройтись в цикле.

2.1. Простейший вариант: range(stop)

Вариант range(stop) создаёт последовательность чисел от 0 до stop-1.

Пример: выведем числа от 0 до 4:

```
for i in range(5):
    print(i)
```

Здесь переменная i последовательно принимает значения 0, 1, 2, 3, 4. Цикл выполнится 5 раз.

2.2. Диапазон со стартом и стопом: range(start, stop)

Вариант range(start, stop) создаёт числа от start до stop-1.

Пример: числа от 3 до 6:

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

Вывод будет: 3, 4, 5, 6. Число 7 не входит, так как верхняя граница всегда не включается.

2.3. Диапазон с шагом: range(start, stop, step)

Третий параметр — шаг изменения значения. Можно считать не по 1, а по 2, по 5 и т.д.

Пример: вывести все нечётные числа от 1 до 9:

```
for i in range(1, 10, 2):
    print(i)
```

Здесь step = 2, поэтому будут значения: 1, 3, 5, 7, 9.

Пример: обратный отсчёт:

```
for i in range(5, 0, -1):
    print(i)
```

Здесь шаг отрицательный: -1. Значения будут: 5, 4, 3, 2, 1.

2.4. Пример: сумма чисел от 1 до 50

Частая задача — посчитать сумму подряд идущих чисел. Сделаем это с помощью цикла for.

```
total = 0 # здесь будем накапливать сумму
for i in range(1, 51): # от 1 до 50 включительно
    total += i

print("Сумма:", total)
```

На каждой итерации к переменной total прибавляется текущий номер i. После завершения цикла в total хранится общая сумма.

2.5. Перебор символов строки

Строка в Python — это последовательность символов, поэтому по ней тоже можно проходить циклом for.

Пример:

```
word = "python"
for ch in word:
    print(ch)
```

Переменная ch по очереди принимает каждый символ строки: 'p', 'y', 't', 'h', 'o', 'n'.

Пример: посчитаем, сколько букв «a» в строке:

```
text = "abracadabra"
count_a = 0

for ch in text:
    if ch == "a":
        count_a += 1

print("Количество букв 'a':", count_a)
```

Мы перебираем каждый символ и, если он равен 'a', увеличиваем счётчик.

3. Цикл while

Цикл while повторяет блок кода, пока условие истинно (True). Если условие сразу ложно, цикл не выполнится ни разу.

3.1. Простой пример: числа от 1 до 10

```
i = 1

while i <= 10:
    print(i)
    i += 1
```

Здесь:
• сначала i = 1;
• перед каждой итерацией проверяется условие i <= 10;
• внутри цикла мы печатаем текущее значение и увеличиваем i на 1;
• как только i станет 11, условие станет ложным, и цикл завершится.

3.2. Обратный отсчёт

```
i = 5

while i > 0:
    print(i)
    i -= 1

print("Старт!")
```

Мы уменьшаем значение переменной i, пока оно больше нуля.

3.3. Ввод чисел до нуля

Частый сценарий: мы читаем числа от пользователя, пока он не введёт специальное значение (например, 0).

```
num = int(input("Введите число (0 — стоп): "))
total = 0

while num != 0:
    total += num
    num = int(input("Введите число (0 — стоп): "))

print("Сумма:", total)
```

Условие num != 0 означает «повторять цикл, пока число не равно нулю». Как только пользователь вводит 0, условие становится ложным, цикл завершается и выводится сумма.

3.4. Цикл while и контроль суммы

Пример: суммируем числа, пока общая сумма меньше 100.

```
total = 0

while total < 100:
    x = int(input("Введите число: "))
    total += x

print("Сумма достигла 100 или больше:", total)
```

4. Цикл while True

Иногда мы заранее не знаем, сколько именно раз нужно повторить действие. В таких случаях удобно использовать конструкцию while True. Это «бесконечный» цикл, из которого мы выходим вручную с помощью команды break (см. ниже).

Пример: запрос команд от пользователя, пока он не введёт 'stop':

```
while True:
    text = input("Введите команду (stop — выход): ")
```

```
    if text == "stop":
        break # выйти из цикла

    print("Вы ввели команду:", text)
```

Условие в while True всегда истинно, поэтому цикл сам по себе не остановится. Остановку мы делаем явно, когда хотим, с помощью break.

5. Условия внутри циклов

Циклы почти всегда используются вместе с условными операторами if / elif / else. Так мы можем по-разному обрабатывать каждую итерацию цикла.

5.1. Пример: чётные и нечётные числа

```
for i in range(1, 11):
    if i % 2 == 0:
        print(i, "— чётное")
    else:
        print(i, "— нечётное")
```

5.2. Пример: подсчёт цифр и букв в строке

```
data = "qwe123asd45"
digits = 0
letters = 0

for ch in data:
    if ch.isdigit():
        digits += 1
    elif ch.isalpha():
        letters += 1

print("Цифр:", digits)
print("Букв:", letters)
```

Здесь мы используем методы строки isdigit() и isalpha(), чтобы определить тип символа.

6. Управление циклом: break, continue, pass

Python даёт три специальных ключевых слова для управления работой цикла:

• break — полностью завершить цикл;

• continue — перейти к следующей итерации, пропустив оставшийся код внутри цикла;

• pass — «ничего не делать», пустая заглушка.

6.1. break — завершить цикл полностью

Команда break используется, когда мы хотим досрочно остановить цикл: например, нашли нужный элемент или пользователь ввёл специальную команду.

Пример: найти первую букву 'a' в слове:

```
word = "kaboom"

for ch in word:
    if ch == "a":
        print("Нашли букву 'a'")
        break # выходим из цикла
    print("Смотрим на символ:", ch)
```

Как только мы встречаем букву 'a', условие выполняется, выводится сообщение, и команда break сразу завершает цикл for. Оставшиеся символы слова уже не будут проверяться.

Пример: игра «угадать число» с использованием while True и break:

```
secret = 7 # загаданное число

while True:
    x = int(input("Угадай число: "))

    if x == secret:
        print("Угадал!")
        break # выходим из цикла

    print("Не угадал, попробуй ещё раз")
```

Здесь цикл будет повторяться, пока пользователь не введёт правильное число. После этого срабатывает break, и цикл завершается.

6.2. continue — перейти к следующей итерации

Команда continue говорит циклу: «этот проход больше не продолжаем, сразу переходи к следующему». Всё, что написано после continue внутри цикла, для текущей итерации выполняться не будет.

Пример: вывести только чётные числа от 1 до 20:

```
for i in range(1, 21):
    if i % 2 != 0: # если число нечётное
        continue    # пропускаем его
    print("Чётное число:", i)
```

Когда i нечётное, условие выполняется, срабатывает continue, и оператор print не выполняется. Для чётных чисел условие ложно, continue не вызывается, и число выводится на экран.

Пример: пропустить пустые строки при вводе:

```
while True:
    text = input("Введите текст ('stop' — выход): ")

    if text == "stop":
        break # завершить цикл

    if text == "": # пустая строка
        continue     # ничего не делаем, переходим к следующему вводу

    print("Вы ввели:", text)
```

Если пользователь просто нажал Enter, переменная text будет пустой строкой. В этом случае мы вызываем continue и не выполняем print, а сразу переходим к следующему запросу ввода.

6.3. pass — пустая заглушка

Ключевое слово pass буквально означает «ничего не делать». Оно используется, когда по синтаксису Python внутри блока должен быть хоть какой-то код, но у нас логики пока нет (например, мы только проектируем программу).

Пример: заглушка внутри цикла:

```
for i in range(5):
    if i == 3:
        pass # здесь позже можно добавить код
    print(i)
```

Пример: заготовка функции:

```
def process_user():
    pass # функция будет реализована позже
```

7. Мини-проект: «Мини-касса»

Рассмотрим небольшой пример, где одновременно используются цикл while True, команды break и continue, проверка условий и работа с числами. Это уже похоже на простую консольную версию кассы в магазине.

Программа должна:

- запрашивать у пользователя цену товара;
- пустой ввод пропускать;
- не принимать отрицательные цены;
- при вводе слова 'pay' завершать работу и показывать итоговую сумму.

Код программы:

```
print("=== Мини касса ===")
print("Вводите цену товара. Пустой ввод — пропуск. 'pay' — оплатить и завершить.")

total = 0 # общая сумма

while True:
    data = input("Цена: ")

    if data == "pay":
        break # выходим из цикла

    if data == "": # пустая строка
        print("Пустой ввод, ничего не добавлено")
        continue # переходим к следующей итерации

    price = float(data) # преобразуем строку в число с точкой

    if price < 0:
        print("Цена не может быть отрицательной")
        continue

    total += price
    print("Текущая сумма:", total)

print("ИТОГО К ОПЛАТЕ:", total)
```

В этой программе мы увидели сразу все ключевые элементы урока:

• цикл while True для многократного ввода;

• break для выхода из цикла по слову 'pay';

• continue для пропуска пустого ввода и некорректных значений;

• работу с числами и строками;

• использование условий внутри цикла.

Эти конструкции — основа для написания интерактивных консольных программ на Python.