Лекция №12. Алгоритмы циклической структуры. Оператор цикла

Содержание:

1. Что такое циклы в Python
2. Зачем нужны циклы в Python
3. Цикл while
4. Цикл for
5. Выход из цикла с помощью break
6. Вложенные циклы
7. Итог

Слайд 2. Циклы в Python — основной инструмент разработчика. С их помощью можно автоматизировать задачи, парой строк кода выполнять несколько действий и генерировать данные.

Циклы в Python, как и в любом другом языке программирования, позволяют выполнять одно действие несколько раз подряд. Конечно, каждую операцию можно отдельно прописать в коде, но это займет много времени и места и такую конструкцию будет тяжело читать. Упростить задачу можно с помощью цикла.

Слайд 3. К примеру, для вывода в консоль цифр от одного до пяти можно воспользоваться пятью отдельными вызовами функции print(). А можно завернуть все в цикл **for**, вызывая функцию print() всего один раз. Вместо пяти строк кода получилось две. Если придется переписать все для вывода чисел с одного до десяти, то надо будет исправить только границы цикла, а не переписывать десять вызовов.

```
for number in range(1, 6):
print(number)
>>> 1
>>> 2
>>> 3
>>> 4
>>> 5
```

Слайд 4. В Python циклы используют для автоматизации процессов как в простых программах, так и в проектах с большой кодовой базой. Обычно циклы применяются:

- **для повторения задач.** С помощью циклов можно вызывать функции несколько раз, а сами циклы могут зависеть от условий;
- **работы с данными.** Можно просуммировать все числа в списке, вывести результат, сохранить его или передать аргументом в другую функцию;
- **итерации.** Цикл **for** позволяет перебрать все символы в строке, подсчитать их количество в общем или по заданному условию — к примеру, только гласные или согласные буквы;
- **обработки файлов.** Можно загрузить документ и с помощью цикла заменить в нем все вхождения определенного слова на другое.

Цикл while

Слайд 5. Цикл **while** позволяет выполнять повторяющиеся действия до тех пор, пока не будет достигнуто определенное условие. Это вид циклов в Python, с которым легче всего разобраться. В общем виде конструкция **while** выглядит следующим образом:

```
while условие:
блок_кода
```

В цикл while входят:

- **условие.** Логическое выражение, определяющее, когда должно прерваться выполнение цикла. Блок кода будет выполняться до тех пор, пока условие истинно;
- **блок кода.** Набор операций и вызовов функций, который запускается при каждой новой итерации.

Слайд 6. Реализуем простейшую проверку ввода корректного пароля пользователем. Для этого создадим переменную password и поместим в нее сам пароль, у нас это будет qwerty. Цикл **while** будет выполняться до тех пор,

пока пользовательское значение будет отличаться от содержимого переменной password.

```
password = "qwerty"
input_password = input("Введите пароль: ")
while input_password != password:
print("❌ Неверный пароль. Попробуйте снова.")
input_password = input("Введите пароль: ")
print("✅ Доступ разрешен. Вы ввели правильный пароль.")
```

Слайд 7. Цикл **while** очень простой, но с ним все равно могут возникать проблемы. К примеру, можно создать бесконечный цикл, который никогда не перестанет выполняться. Его смогут прервать только принудительное завершение программы, переполнение памяти или смена условия в коде.

```
while True:
print("Это бесконечный цикл.")
```

В примере выше условие **True** всегда истинно и не изменяется, а значит, ничто не сможет завершить выполнение блока кода. В результате получим бесконечный вывод строки в консоль. Следует внимательно следить за тем, чтобы всегда было условие выхода из цикла.

Цикл for

Слайд 8. С помощью **for** в Python удобно выполнять повторяющиеся действия для каждого элемента в коллекции. Количество повторений зависит от того, сколько элементов надо пройти, а выход из цикла происходит после того, как элементы для итерирования закончились. В общем виде цикл **for** выглядит так:

```
итерируемая_коллекция = [..., ..., ...]
for временная_переменная in итерируемая_коллекция:
блок_кода
```

В него входят:

- **итерируемая коллекция.** Структура данных, по которой проходит цикл;

- **временная переменная.** Нужна для работы со значениями коллекции на каждом новом шаге выполнения цикла;
- **блок кода.** Действия с переменной, выполняемые на каждом круге.

Слайд 9. Теперь напишем цикл, который проходится по набору чисел и определяет, четные они или нет. Будем использовать **for** и условия внутри блока кода.

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]
for num in numbers:
if num % 2 == 0:
print(f"{num} - четное число")
else:
print(f"{num} - нечетное число")
>>> 1 - нечетное число
>>> 2 - четное число
>>> 3 - нечетное число
>>> 4 - четное число
>>> 5 - нечетное число
>>> 6 - четное число
>>> 7 - нечетное число
>>> 8 - четное число
>>> 9 - нечетное число
```

Выход из цикла с помощью break

Слайд 10. Иногда из цикла надо выйти до того, как он закончит выполняться, — к примеру, при достижении определенного условия, влияющего на дальнейшую логику программы. В Python это можно сделать с помощью ключевого слова **break**.

Представим, что надо пройтись по всем числам в коллекции с помощью цикла **for**, но выйти из него, если встретилась единица.

```
numbers = [2, 3, 4, 1, 5, 6, 7, 8, 9]
for num in numbers:
```

```
if num == 1:
print("Выход из цикла")
break
print(num)
>>> 2
>>> 3
>>> 4
>>> Выход из цикла
```

Выполнение цикла прервалось до того, как программа успела пройтись по всем элементам коллекции. Все из-за того, что мы достигли условия и сработало ключевое слово **break**. Если убрать единицу из коллекции, то цикл выведет все числа.

Вложенные циклы

В Python можно из одного цикла вызывать другой цикл. В таком случае код усложняется, но это полезно во многих задачах. К примеру, надо пройтись по всем значениям таблицы. Одним циклом перебираем значения ячеек в строке, а вторым — переключаемся на новую строку. Общая запись вложенных циклов выглядит так:

```
for условие:
блок_кода
for условие:
блок_кода
```

Слайд 12. Для примера представим, что у нас есть таблица из двух строк и четырех столбцов. Обойдем каждую ее ячейку с помощью вложенного цикла, выводя индекс каждой ячейки.

```
for i in range(1, 3):
for j in range(1, 5):
print(f'Ячейка {i}:{j}')
>>> Ячейка 1:1
>>> Ячейка 1:2
```

```
>>> Ячейка 1:3
>>> Ячейка 1:4
>>> Ячейка 2:1
>>> Ячейка 2:2
>>> Ячейка 2:3
>>> Ячейка 2:4
```

Итог

- Циклы — базовый инструмент программирования на Python. С их помощью разработчики могут быстро выполнять повторяющиеся действия, автоматизируя процесс.
- В каждом цикле есть условие и блок кода, которые выполняются, пока условие истинно.
- С помощью циклов for можно итерировать коллекции данных, выполняя преобразования.
- В циклах while задается явное условие.
- С помощью ключевого слова break можно в любой момент прервать выполнение цикла.