ПМ2 Прикладное программирование.

РО 2.1. Создавать консольные приложения на Python.

Тема 7. Цикл for.

Цель занятия:

Сформировать понимание конструкции цикла for в Python.

Учебные вопросы:

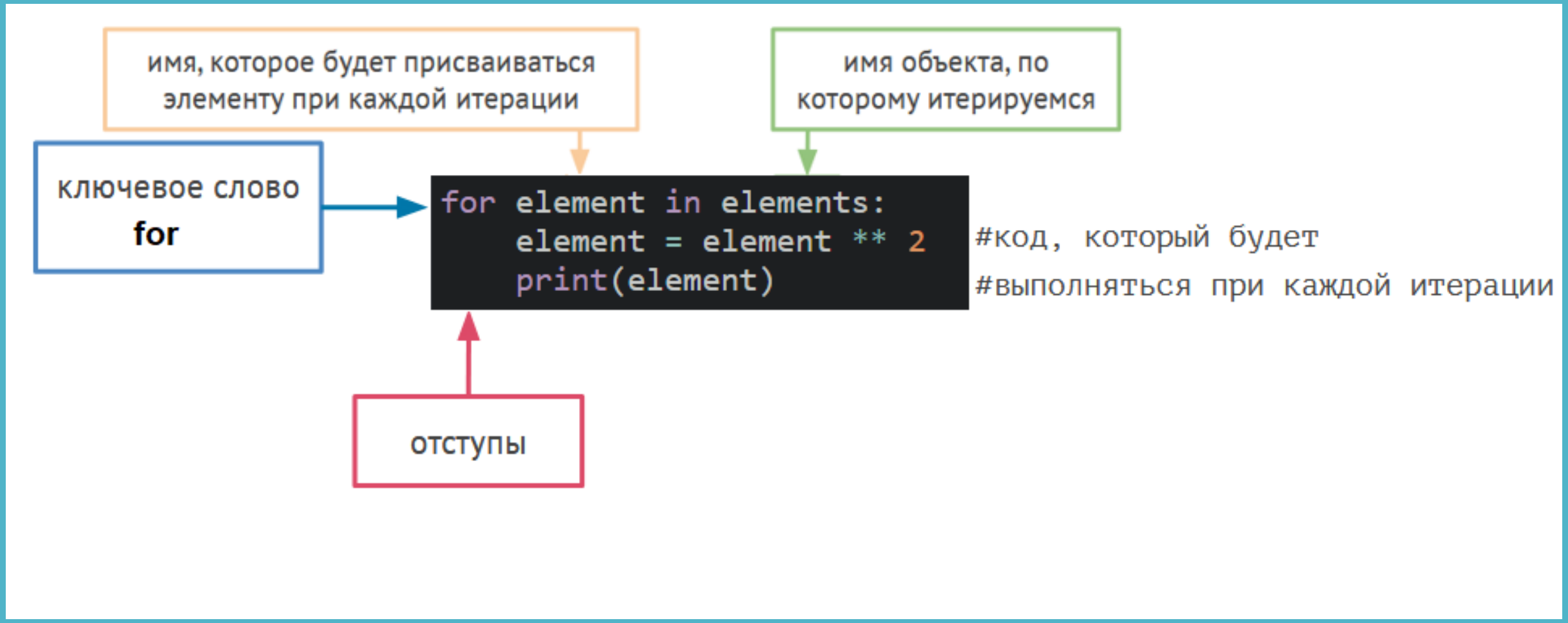
1. Цикл for в Python
2. Функция range()
3. Функция enumerate()
4. Операторы break, continue и pass в цикле for

1. Цикл for в Python

Цикл **for** — это управляющая конструкция, позволяющая последовательно перебрать все элементы некоторой итерируемой последовательности (списка, строки, словаря, множества, диапазона чисел и т. д.).

В отличие от некоторых других языков, где цикл **for** основан на счётчике, в Python он основан на итерации по готовой последовательности.

Синтаксис:


имя, которое будет присваиваться элементу при каждой итерации
имя объекта, по которому итерируемся
ключевое слово for
for element in elements:
element = element ** 2
print(element)
#код, который будет
#выполняться при каждой итерации
отступы

переменная — имя, в которое по очереди попадают элементы последовательности.

последовательность — список объектов, по которым осуществляется перебор. Это может быть список, строка, словарь, множество или объект range().

тело_цикла — блок команд, выполняющийся для каждого элемента.

Переменная цикла автоматически принимает значение очередного элемента из последовательности, создаётся внутри цикла, может называться произвольно.

Пример:

```
for fruit in ["яблоко", "груша", "слива"]:
    print("Фрукт:", fruit)
```

Здесь **fruit** — переменная цикла. Каждый проход цикла она получает новое значение из списка.

Важно:

- Имя переменной цикла желательно выбирать осмысленным.
- После завершения цикла переменная продолжает существовать и хранит последнее значение.

Принцип итерации по элементам последовательности.

Итерация — это процесс последовательного перебора элементов **коллекции**.

Цикл **for**:

- берёт первый элемент,
- присваивает его переменной цикла,
- выполняет тело цикла,
- затем берёт следующий элемент,
- повторяет до исчерпания элементов.
- Если в последовательности 0 элементов, тело цикла не выполнится ни разу.

Итерация по различным типам данных.

1) Список

```
numbers = [10, 20, 30]
for n in numbers:
    print(n)
```

Каждый элемент списка поступает в переменную цикла.

2) Строка

Строка — тоже последовательность. Итерация идёт по символам:

```
for char in "Привет":
    print(char)
```

Выводится один символ на каждой строке.

Цикл for — один из ключевых инструментов Python, позволяющий работать с любыми итерируемыми объектами.

Он удобен тем, что автоматически берёт следующий элемент, не требуя ручного управления индексами (как в других языках).

Благодаря своей гибкости он используется практически во всех программах.

2. Функция range()

Функция **range()** в Python создаёт специальный объект, представляющий последовательность целых чисел.

Этот объект не является полноценным списком, но может использоваться в циклах **for** или преобразовываться в список при необходимости.

range() удобен для:

- повторения действий определённое число раз;
- генерации последовательностей чисел;
- работы с индексами списка;
- создания шагов в циклах.

Особенности объекта **range**:

- range не хранит все числа в памяти, а генерирует их по мере обращения — это экономит ресурсы.
- Значения в range изменяются только целыми числами.
- Может задаваться одним, двумя или тремя параметрами.

Формы записи **range()**

1) **range(stop)**

Создаёт последовательность чисел от 0 до stop - 1.

Пример:

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

Вывод:

0
1
2
3
4

2) **range(start, stop)**

Числа создаются от start до stop - 1.

```
for i in range(2, 6):
    print(i)
```

Вывод:

2

3

4

5

3) **range(start, stop, step)**

Третий параметр — шаг изменения.

```
for i in range(0, 10, 2):
    print(i)
```

Вывод:

0

2

4

6

8

Шаг может быть отрицательным:

```
for i in range(10, 0, -2):
    print(i)
```

Вывод:

10
8
6
4
2

Так можно организовать обратный отсчёт.

Функция **range()** — основной инструмент создания последовательностей чисел в Python и один из фундаментальных элементов при работе с циклами **for**.

Она позволяет компактно и эффективно задавать диапазоны, контролировать начало, конец и шаг перебора, а также избегать лишних вычислений благодаря ленивой генерации.

3. Функция enumerate()

Функция enumerate() применяется при переборе последовательностей, когда необходимо одновременно получать:

- индекс элемента,
- сам элемент.

Синтаксис функции:

enumerate(последовательность, start=0)

Параметры:

- последовательность — объект, по которому выполняется итерация (список, строка, кортеж и т.п.)
- start — начальное значение индекса (по умолчанию 0)

Принцип работы.

enumerate() преобразует последовательность в набор пар:

- (индекс, значение)

При каждом проходе цикла for переменные получают:

- номер элемента,
- сам элемент.

Простой пример со списком:

```
fruits = ["яблоко", "груша", "слива"]

for index, fruit in enumerate(fruits):
    print(index, fruit)
```

Результат:

0 яблоко

1 груша

2 слива

Изменение стартового индекса

Можно указать начало счёта не с 0, а с 1 или другого числа:

```
fruits = ["яблоко", "груша", "слива"]

for i, fruit in enumerate(fruits, start=1):
    print(i, fruit)
```

Результат:

1 яблоко

2 груша

3 слива

Что возвращает **enumerate()**?

Не список, а специальный итератор.

Чтобы увидеть все пары, можно преобразовать в список:

```
fruits = ["яблоко", "груша", "слива"]

ls = list(enumerate(fruits))

print(ls)
```

Результат:

[(0, 'яблоко'), (1, 'груша'), (2, 'слива')]

4. Операторы break, continue и pass в цикле for

В цикле for (и while) используются специальные операторы, которые позволяют менять стандартный ход выполнения цикла.

К таким операторам относятся:

- **break** — завершает цикл полностью.
- **continue** — пропускает текущую итерацию.
- **pass** — ничего не делает (заглушка), используется для синтаксического заполнения блока.

Оператор **break**

Полностью прекращает выполнение цикла, даже если последовательность ещё не закончилась.

При выполнении break интерпретатор выходит из данного цикла и переходит к коду после него.

Пример:

```
for n in [1, 2, 3, 4, 5]:
    if n == 3:
        break
    print(n)
```

Вывод:

1

2

Оператор **continue**

Пропускает выполнение оставшейся части тела цикла только для текущей итерации и переходит к следующему элементу.

Когда встречается continue, интерпретатор мгновенно переходит к следующему шагу цикла.

Пример:

```
for n in [1, 2, 3, 4, 5]:
    if n == 3:
        continue
    print(n)
```

Вывод:

1

2

4

5

Оператор **pass**

Ничего не делает — выполняется как пустая операция.

Используется, когда синтаксис требует наличие инструкции, но её логика ещё не реализована.

Пример:

```
for n in [1, 2, 3]:
    if n == 2:
        pass  # пока ничего не делаем
    print("Обрабатываем:", n)
```

Вывод:

Обрабатываем: 1

Обрабатываем: 2

Обрабатываем: 3

Итог:

- break — прерывает цикл полностью.
- continue — пропускает текущую итерацию.
- pass — пустая операция, используется как заглушка.

Эти три оператора дают возможность гибко управлять логикой выполнения циклов и часто используются в реальных программах.

Контрольные вопросы:

- Для чего в программировании используются циклы?
- Чем цикл for отличается от цикла while?
- Как выглядит общий синтаксис цикла for в Python?
- Что такое «итерация»? Что такое «итераторная последовательность»?
- Как работает функция range() и какие формы записи она поддерживает?
- Что делает переменная в конструкции for x in ...?
- Можно ли перебирать символы строки с помощью цикла for?
- Для чего используется функция enumerate()?
- Чем отличается break от continue в цикле for?

Домашнее задание:

https://ru.hexlet.io/programs/python-basics-free

Материалы лекций:

https://github.com/ShViktor72/education2025

ПМ2_Прикладное_программирование

Обратная связь:

colledge20education23@gmail.com