**Информация об уроке**

**Модуль: 1**

**Номер занятия: 10**

**Тип занятия: Лекция**

**Название занятия/ Тема занятия:** Циклы; операторы Break, Continue; Массивы

**Цель занятия:** изучить базовые настройки Python.

**Образовательные результаты:** Изучить переменные, их типы данных и базовые операторы.

**Глоссарий:**

**Конспект занятия**

1. **Приветствие**

Приветствуем вас на новом курсе по Python!

1. **Тема урока и целеполагание**

Сегодня мы пройдём:

1. Циклы;
2. операторы Break, Continue;
3. Массивы

Эти темы также входят в базовые во всех языках программирования, не только в Python.

1. **Актуализация**

Для начала вспомним материал прошлого урока:

Переменная хранит данные одного из типов данных:

* **int** (integer) – число
* **float** (плавающая точка) – дробное число
* **str** (string) – строка
* **bool** (булева функция) – True или False (правда или ложь / из двоичной логики)

**Преобразование типа данных**

Для изменения типа данных используется следующая конструкция:



Python поддерживает все распространенные арифметические операции:

* + сложение
* - вычитание
* = присваивание
* \* умножение
* / деление
* \*\* возведение в степень
* // целочисленное деление
* % остаток от деления

Ряд операций представляют условные выражения. Все эти операции принимают два операнда и возвращают логическое значение.

Простейшие условные выражения представляют операции сравнения, которые сравнивают два значения. Python поддерживает следующие операции сравнения:

| **Оператор** | **Описание** | **Пример** |
| --- | --- | --- |
| == | Возвращает True, если оба операнда равны | a == b |
| != | Возвращает True, если оба операнда не равны | a != b |
| > | Возвращает True, если первый операнд больше второго | a > b |
| < | Возвращает True, если первый операнд меньше второго | a < b |
| >= | Возвращает True, если первый операнд больше или равен второму | a >= b |
| <= | Возвращает True, если первый операнд меньше или равен второму | a <= b |

Для создания составных условных выражений применяются логические операции. В Python имеются следующие логические операторы:

**and (логическое умножение)**

Возвращает True, если оба выражения равны True

**or (логическое сложение)**

Возвращает True, если хотя бы одно из выражений равно True

**not (логическое отрицание)**

Возвращает True, если выражение равно False

**Оператор in**

Оператор in возвращает True если в некотором наборе значений есть определенное значение. Он имеет следующую форму.

Например, строка представляет набор символов. И с помощью оператора in мы можем проверить, есть ли в ней какая-нибудь подстрока.

Если нам надо наоборот проверить, нет ли в наборе значений какого-либо значения, то мы можем использовать модификацию оператора - **not in**. Она возвращает True, если в наборе значений НЕТ определенного значения:

Условные конструкции используют условные выражения и в зависимости от их значения направляют выполнение программы по одному из путей. Одна из таких конструкций - это конструкция **if**, если это логическое выражение возвращает True, то выполняется последующий блок инструкций, каждая из которых должна начинаться с новой строки и должна иметь отступы от начала выражения if (отступ необходимо делать в 4 пробела).

Если вдруг нам надо определить альтернативное решение на тот случай, если выражение в if возвратит False, то мы можем использовать блок **else**.

Если необходимо ввести несколько альтернативных условий, то можно использовать дополнительные блоки **elif**, после которого идет блок инструкций.

Сначала Python проверяет выражение if. Если оно равно True, то выполняются инструкции из блока if. Если это условие возвращает False, то Python проверяет выражение из elif.

Если выражение после elif равно True, то выполняются инструкции из блока elif. Но если оно равно False то выполняются инструкции из блока else

При необходимости можно определить несколько блоков elif для разных условий. Например:

Конструкция **if** в свою очередь сама может иметь вложенные конструкции if (условия в условии).

Отладкой кода называют процесс последовательного приведения кода в рабочее состояние после нахождения ошибки. Существуют различные методы отладки:

* **Debug** - выполнение программы в режиме действие за действием, чтобы напрямую вычислить ошибку (доступно только в профессиональных средах программирования)
* **Проверки выводом** - добавление функций print() в разные части программы, чтобы видеть состояния переменных и программы в тот или иной момент времени выполнения.

Следующей не менее важной управляющей конструкцией в программировании являются **циклы**. Если брать аналогию из жизни - набор действий, который повторяется из раза в раз. Отличие Python от жизни в том, что этот набор действий повторяется до тех пор, пока не прервется условие повторения. Циклы позволяют перебирать списки, выполнять сортировки, создавать игры и сложные программы.

1. **Основное содержание**

**1. Циклы.**

Циклы позволяют выполнять некоторое действие в зависимости от соблюдения некоторого условия. В языке Python есть следующие типы циклов:

* **for**
* **while**

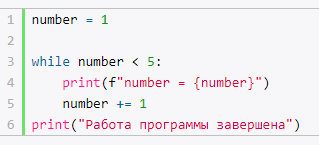
**Цикл while**

Цикл while проверяет истинность некоторого условия, и если условие истинно, то выполняет инструкции цикла. Он имеет следующее формальное определение:



После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала ключевого слова while.



В данном случае цикл while будет выполняться, пока переменная number меньше 5.

Сам блок цикла состоит из двух инструкций:



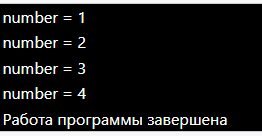
Обратите внимание, что они имеют отступы от начала оператора while - в данном случае от начала строки. Благодаря этому Python может определить, что они принадлежат циклу. В самом цикле сначала выводится значение переменной number, а потом ей присваивается новое значение. .

Также обратите внимание, что последняя инструкция print("Работа программы завершена") не имеет отступов от начала строки, поэтому она не входит в цикл while.

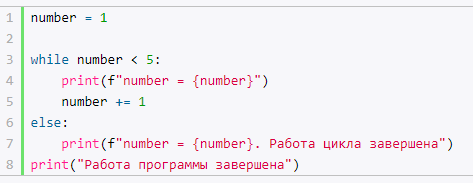
Весь процесс цикла можно представить следующим образом:

1. Сначала проверяется значение переменной number - больше ли оно 5. И поскольку вначале переменная равна 1, то это условие возвращает True, и поэтому выполняются инструкции цикла
2. Инструкции цикла выводят на консоль строку number = 1. И далее значение переменной number увеличивается на единицу - теперь она равна 2. Однократное выполнение блока инструкций цикла называется **итерацией**. То есть таким образом, в цикле выполняется первая **итерация**.
3. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 2, поэтому выполняются инструкции цикла
4. Инструкции цикла выводят на консоль строку number = 2. И далее значение переменной number опять увеличивается на единицу - теперь она равна 3. Таким образом, выполняется вторая итерация.
5. Опять проверяется условие number < 5. Оно по прежнему равно True, так как number = 3, поэтому выполняются инструкции цикла
6. Инструкции цикла выводят на консоль строку number = 3. И далее значение переменной number опять увеличивается на единицу - теперь она равна 4. То есть выполняется третья итерация.
7. Снова проверяется условие number < 5. Оно по прежнему равно True, так как number = 4, поэтому выполняются инструкции цикла
8. Инструкции цикла выводят на консоль строку number = 4. И далее значение переменной number опять увеличивается на единицу - теперь она равна 5. То есть выполняется четвертая итерация.
9. И вновь проверяется условие number < 5. Но теперь оно равно False, так как number = 5, поэтому выполняются выход из цикла. Все цикл - завершился. Дальше уже выполняются действия, которые определены после цикла. Таким образом, данный цикл произведет четыре прохода или четыре итерации

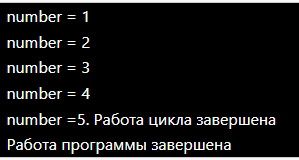
В итоге при выполнении кода мы получим следующий консольный вывод:



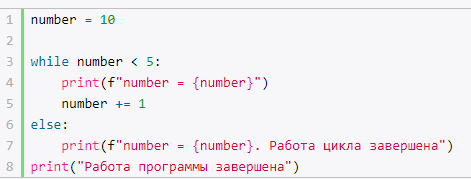
Для цикла while также можно определить дополнительный блок **else**, инструкции которого выполняются, когда условие равно False:



То есть в данном случае сначала проверяется условие и выполняются инструкции while. Затем, когда условие становится равным False, выполняются инструкции из блока else. Обратите внимание, что инструкции из блока else также имеют отступы от начала конструкции цикла. В итоге в данном случае мы получим следующий консольный вывод:



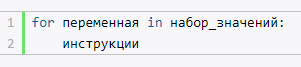
Блок else может быть полезен, если условие изначально равно False, и мы можем выполнить некоторые действия по этому поводу:



В данном случае условие number < 5 изначально равно False, поэтому цикл не выполняет ни одной итерации и сразу переходит в блоку else.

**Цикл for**

Другой тип циклов представляет конструкция for. Этот цикл пробегается по набору значений, помещает каждое значение в переменную, и затем в цикле мы можем с этой переменной производить различные действия. Формальное определение цикла for:

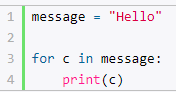


После ключевого слова for идет название переменной, в которую будут помещаться значения. Затем после оператора in указывается набор значений и двоеточие.

А со следующей строки располагается блок инструкций цикла, которые также должны иметь отступы от начала цикла.

При выполнении цикла Python последовательно получает все значения из набора и передает их в переменную. Когда все значения из набора будут перебраны, цикл завершает свою работу.

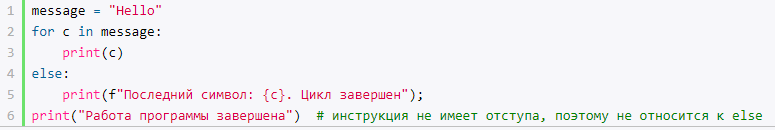
В качестве набора значений, например, можно рассматривать строку, которая по сути представляет набор символов. Посмотрим на примере:



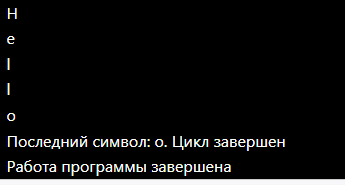
В цикле определяется переменную c, после оператора in в качестве перебираемого набора указана переменная message, которая хранит строку "Hello". В итоге цикл for будет перебирать последовательно все символы из строки message и помещать их в переменную c. Блок самого цикла состоит из одной инструкции, которая выводит значение переменной с на консоль. Консольный вывод программы:



Цикл for также может иметь дополнительный блок else, который выполняется после завершения цикла:



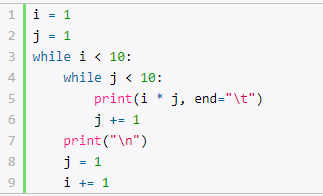
В данном случае мы получим следующий консольный вывод:



Стоит отметить, что блок else имеет доступ ко всем переменным, которые определены в цикле for.

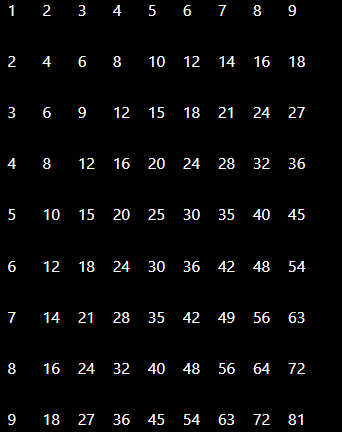
**Вложенные циклы**

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения:

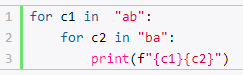


Внешний цикл while i < 10: срабатывает 9 раз пока переменная i не станет равна 10. Внутри этого цикла срабатывает внутренний цикл while j < 10:. Внутренний цикл также срабатывает 9 раз пока переменная j не станет равна 10. Причем все 9 итераций внутреннего цикла срабатывают в рамках одной итерации внешнего цикла.

В каждой итерации внутреннего цикла на консоль будет выводится произведение чисел i и j. Затем значение переменной j увеличивается на единицу. Когда внутренний цикл закончил работу, значений переменной j сбрасывается в 1, а значение переменной i увеличивается на единицу и происходит переход к следующей итерации внешнего цикла. И все повторяется, пока переменная i не станет равна 10. Соответственно внутренний цикл сработает всего 81 раз для всех итераций внешнего цикла. В итоге мы получим следующий консольный вывод:



Подобным образом можно определять вложенные циклы for:



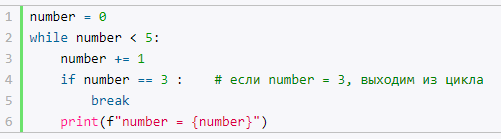
В данном случае внешний цикл проходит по строке "ab" и каждый символ помещает в переменную c1. Внутренний цикл проходит по строке "ba", помещает каждый символ строки в переменную c2 и выводит сочетание обоих символов на консоль. То есть в итоге мы получим все возможные сочетания символов a и b:



**Выход из цикла. break и continue**

Для управления циклом мы можем использовать специальные операторы break и continue. Оператор break осуществляет выход из цикла. А оператор continue выполняет переход к следующей итерации цикла.

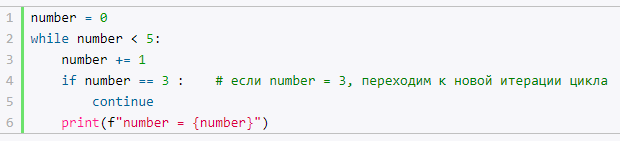
Оператор break может использоваться, если в цикле образуются условия, которые несовместимы с его дальнейшим выполнением. Рассмотрим следующий пример:



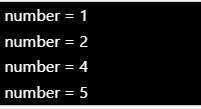
Здесь цикл while проверяет условие number < 5. И пока number не равно 5, предполагается, что значение number будет выводиться на консоль. Однако внутри цикла также проверяется другое условие: if number == 3. То есть, если значение number равно 3, то с помощью оператора break выходим из цикла. И в итоге мы получим следующий консольный вывод:



В отличие от оператора break оператор continue выполняет переход к следующей итерации цикла без его завершения. Например, в предыдущем примере заменим break на continue:



И в этом случае если значение переменной number равно 3, последующие инструкции после оператора continue не будут выполняться:



**2.Массивы.**

Массив (Список или list) представляет тип данных, который хранит набор или последовательность элементов. Во многих языках программирования есть аналогичная структура данных, которая называется массив.

**Создание списка**

Для создания списка применяются квадратные скобки **[]**, внутри которых через запятую перечисляются элементы списка. Например, определим список чисел:



Подобным образом можно определять списки с данными других типов, например, определим список строк:



Также для создания списка можно использовать функцию-конструктор **list()**:

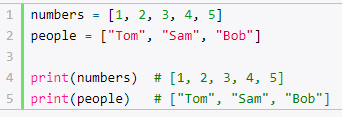


Оба этих определения списка аналогичны - они создают пустой список.

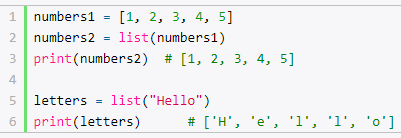
Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных:



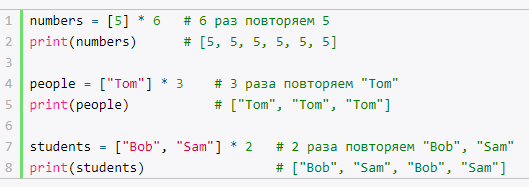
Для проверки элементов списка можно использовать стандартную функцию print, которая выводит содержимое списка в удобочитаемом виде:



Конструктор list может принимать набор значений, на основе которых создается список:

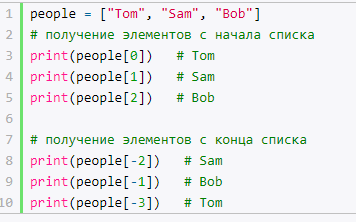


Если необходимо создать список, в котором повторяется одно и то же значение несколько раз, то можно использовать символ звездочки \*, то есть фактически применить операцию умножения к уже существующему списку:

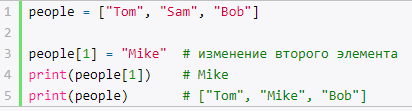


**Обращение к элементам списка**

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.



Для изменения элемента списка достаточно присвоить ему новое значение:



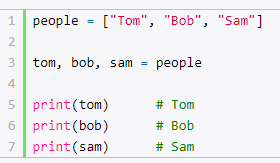
Индексация начинается с 0 из-за того, что имя массива – это фактически указатель на первый его элемент, тип данных определяет на сколько байт делать отступ от первого элемента.

То есть a[1] = a[0] + 1\*n байт, n байт – количество памяти, выделяемое на определенный тип данных. Первый элемент в сдвиге не нуждается, поэтому его номер – 0, указатель на первый элемент + 0 байт.



**Разложение списка**

Python позволяет разложить список на отдельные элементы:

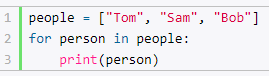


В данном случае переменным tom, bob и sam последовательно присваиваются элементы из списка people. Однако следует учитывать, что количество переменных должно быть равно числу элементов присваиваемого списка.

**Перебор элементов**

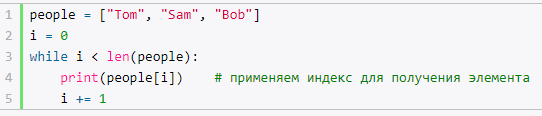
Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла **for**:



Здесь будет производиться перебор списка people, и каждый его элемент будет помещаться в переменную person.

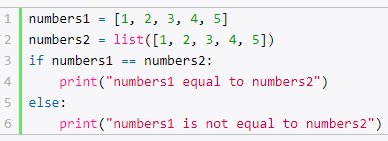
Перебор также можно сделать с помощью цикла **while**:



Для перебора с помощью функции **len()** получаем длину списка. С помощью счетчика i выводит по элементу, пока значение счетчика не станет равно длине списка.

**Сравнение списков**

Два списка считаются равными, если они содержат один и тот же набор элементов:



В данном случае оба списка будут равны.

1. **Подведение итогов/Рефлексия**

На этом занятии мы узнали что такое циклы, каких видов они бывают. Что делают операторы break, continue. Также мы изучили массивы и методы работы с ними.