**Информация об уроке**

**Модуль: 1**

**Номер занятия: 8**

**Тип занятия: Лекция**

**Название занятия/ Тема занятия:** Условные выражения; Условные конструкции; Отладка кода

**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. **Условные выражения**
2. **Отладка кода**
3. **Актуализация**

Вспомним основные моменты из прошлого урока:

**Переменные** предназначены для хранения данных.

В Python применяется два типа наименования переменных: **camel case** и **underscore notation**.

**Camel case** подразумевает, что каждое новое подслово в наименовании переменной начинается с большой буквы. Например:

**Underscore notation** подразумевает, что подслова в наименовании переменной разделяются знаком подчеркивания. Например:

Определив переменную, мы можем использовать её в программе.

Отличительной особенностью переменной является то, что мы можем менять ее значение в течение работы программы:

Компьютеру необходимо заранее знать какого рода данные (тип данных) будут храниться в той или иной области памяти (переменной) - целое число, дробное число, логическая функция, текст или комплексное число. Это нужно для того, чтобы ЭВМ понимала какие операции и действия мы можем совершать над переменной.

**Тип данных** – описание того, какого рода данные хранятся в переменной.

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)

Строка может содержать ряд специальных символов - **управляющих последовательностей**. Некоторые из них:

* **\** - позволяет добавить внутрь строки слеш
* **\' -**  позволяет добавить внутрь строки одинарную кавычку
* **\" -**  позволяет добавить внутрь строки двойную кавычку
* **\n -**  осуществляет переход на новую строку
* **\t -**  добавляет табуляцию (4 отступа)

Python позволяет встравивать в строку значения других переменных. Для этого внутри строки переменные размещаются в фигурных скобках **{}**, а перед всей строкой ставится символ **f**:  
Python является языком с динамической типизацией. А это значит, что переменная не привязана жестко с определенному типу.

Тип переменной определяется исходя из значения, которое ей присвоено.

При этом в процессе работы программы мы можем изменить тип переменной, присвоив ей значение другого типа:

С помощью встроенной функции **type()** динамически можно узнать текущий тип переменной:

Для изменения типа данных используется следующая конструкция:



a = **int(**input()**)** или a = **float(**input()**)**

Python поддерживает все распространенные арифметические операции:

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

**ПРИМЕЧАНИЕ!**

Также возможно сложение строк: **str + str**; и умножение строки на число: **str \* int**:

**Операции и их направление**

**\*\*** - Справа налево

**\* / // %** - Слева направо

**+ -** - Слева направо

Можно заметить, что одни лишь переменные и арифметические операции дают довольно скудный набор инструментов для решения задач. В языка программирования Python существуют также и другие инструменты, такие как: условные выражения, циклы, функции, классы.

Сегодня мы с вами углубимся в изучение условных выражений.

Даже в жизни мы часто ставим условия, того не замечая.

Пример: если я получу 5 - я буду чувствовать себя хорошо. Условие задает нужный вектор для эмоционального состояния человека. Программа - тот же организм, которому мы можем задавать направление действий.

1. **Основное содержание**

**1. Условные выражения.**

Программа на языке Python состоит из набора инструкций. В основе инструкций могут использоваться операторы.

Оператор в Python — это символ, который выполняет операцию над одним или несколькими операндами.

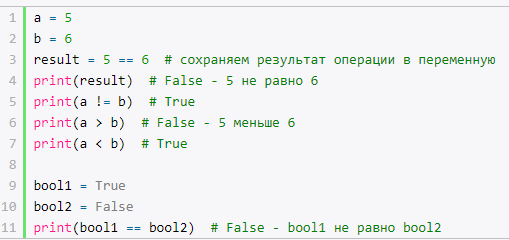
Ряд операций представляют условные выражения. Все эти операции принимают два операнда и возвращают логическое значение, которое в Python представляет тип bool. Существует только два логических значения - True (выражение истинно) и False (выражение ложно).

**Операции сравнения**

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 |

Примеры операций сравнения:



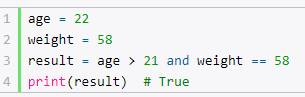
Операции сравнения могут сравнивать различные объекты - строки, числа, логические значения, однако оба операнда операции должны представлять один и тот же тип.

**Логические операции**

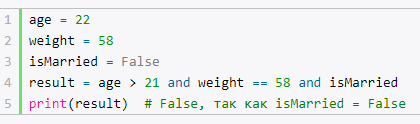
Для создания составных условных выражений применяются логические операции. В Python имеются следующие логические операторы:

**and (логическое умножение)**

Возвращает True, если оба выражения равны True

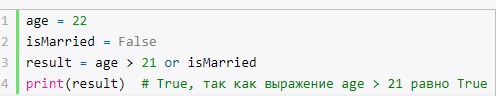


В данном случае оператор and сравнивает результаты двух выражений: age > 21 weight == 58. И если оба этих выражений возвращают True, то оператор and также возвращает True. Причем в качестве одно из выражений необязательно выступает операция сравнения: это может быть другая логическая операция или просто переменная типа boolean, которая хранит True или False.



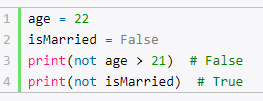
**or (логическое сложение)**

Возвращает True, если хотя бы одно из выражений равно True



**not (логическое отрицание)**

Возвращает True, если выражение равно False



Если один из операндов оператора and возвращает False, то другой операнд уже не оценивается, так как оператор в любом случае возвратит False. Подобное поведение позволяет немного увеличить производительность, так как не приходится тратить ресурсы на оценку второго операнда.

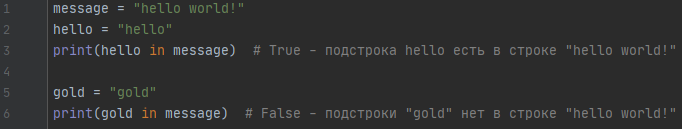
Аналогично если один из операндов оператора or возвращает True, то второй операнд не оценивается, так как оператор в любом случае возвратит True.

**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:

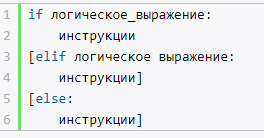


Если нам надо наоборот проверить, нет ли в наборе значений какого-либо значения, то мы можем использовать модификацию оператора - **not in**. Она возвращает True, если в наборе значений НЕТ определенного значения:

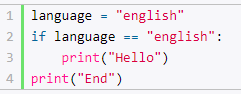


**2. Условная конструкция if – elif – else.**

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**. Она имеет следующее формальное определение:

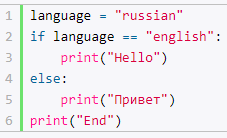


В самом простом виде после ключевого слова **if** идет логическое выражение. И если это логическое выражение возвращает True, то выполняется последующий блок инструкций, каждая из которых должна начинаться с новой строки и должна иметь отступы от начала выражения if (отступ необходимо делать в 4 пробела или то количество пробелов, которое кратно 4):



**Блок else**

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

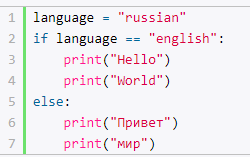


Если выражение language == "english" возвращает True, то выполняется блок if, иначе выполняется блок else. И поскольку в данном случае условие language == "english" возвращает False, то будут выполняться инструкция из блока else.

Причем инструкции блока **else** также должны имеет отступы от начала строки. Например, в примере выше print("End") не имеет отступа, поэтому она не входит в блок else и будет выполняться вне зависимости, чему равно условие language == "english". То есть консоль нам выведет следующие строки:



Блок else также может иметь несколько инструкций, которые должны иметь отступ от начала строки:



**elif**

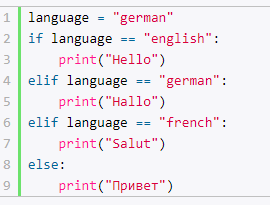
Если необходимо ввести несколько альтернативных условий, то можно использовать дополнительные блоки elif, после которого идет блок инструкций.



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.

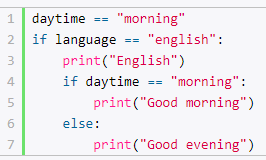
Если выражение после elif равно True, то выполняются инструкции из блока elif. Но если оно равно False то выполняются инструкции из блока else

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



**Вложенные конструкции if**

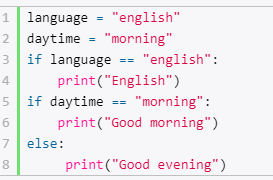
Конструкция if в свою очередь сама может иметь вложенные конструкции if:



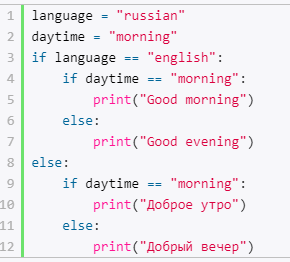
Здесь конструкция if содержит вложенную конструкцию if/else. То есть если переменная language равна "english", тогда вложенная конструкция if/else дополнительно проверяет значение переменной daytime - равна ли она строке "morning" ли нет. И в данном случае мы получим следующий консольный вывод:



Стоит учитывать, что вложенные выражения if также должны начинаться с отступов, а инструкции во вложенных конструкциях также должны иметь отступы. Отступы, расставленные не должным образом, могут изменить логику программы. Так, предыдущий пример НЕ аналогичен следующему:



Подобным образом можно размещать вложенные конструкции if/elif/else в блоках elif и else:



**2. Отладка кода**

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.

1. **Подведение итогов/Рефлексия**

На этом занятии мы узнали что такое условие, как оно формируется. Отличия условного выражения от условной конструкции. Блоки if - elif - else. Что такое отладка кода.