**Уроки Python для начинающих | #1 - Программирование на Python**

Уроки программирования Python для начинающих. Обучение с нуля для чайников. Большой видео курс по изучению языка Питон.

Питон входит в топ популярнейших языков программирования по нынешним оценкам рейтинга TIOBE. Именно с него многие программисты начинают карьеру в своих домах или университетах. Вам не составит особого труда самостоятельно разобраться в языке. Программирование на нём приносит эстетическое удовольствие и неплохой доход. Даже у неопытного программиста здесь получается лаконичный и легко читаемый код. Питон – это красивый и очень мощный язык.

Из-за распространения языка, вокруг него ходит много легенд, мифов и тайн. Многие начинающие программисты просто не понимают, почему Python невероятно полезен в современном мире. По этой причине и решили составить небольшой обзорный материал.

Где применяется Python?

Несколько основных функциональных достоинств Python:

Умеет работать с расширениями xml/html;

Поддерживает управление http-запросами;

Обладает графическим интерфейсом;

Помогает создавать веб-скрипты;

Может работать с FTP;

Способен оперировать картинками, видео и звуковыми файлами;

Используется в разработке робототехники;

Отвечает за создание научных, вычислительных площадок и ещё много всего другого.

Можно сказать, что Python отлично подходит для выполнения большей части ежедневным задач программистов. Участвует как в создании обычных бекапов или чтении писем с почты, так и в разработке крупных игр. Питон практически не имеет ограничений по сферам использования, что позволяет использовать его в проектах колоссальных масштабов. Чтобы не быть голословными, Python часто применяется китами мира IT – Google и Яндекс. Если добавить к этому простоту создания программ, несложно понять, почему Питон заслуживает место в списке лучших языков.

Это невероятно мощный язык программирования, который может выучить даже школьник, если будет им заниматься после занятий. Чтобы начать работать на Python, прежде стоит пройти на официальный веб-ресурс для скачивания дистрибутива языка. На сайте есть масса полезной информации в отношении правил написания кода и расширяемости языка.

Сфера применения языка выходит далеко за пределы разработки обычных приложений. Он используется в разработке игр, веб-программировании и многом другом. Для работы в подобных направлениях нужно только подобрать, скачать и установить соответствующий фреймворк. Эта библиотека поможет расширить функции языка в конкретном направлении. С фреймворками проще создавать проекты любых масштабов.

На сегодня популярнейшей библиотекой является Django - это фреймворк для веб-разработки сайтов, в котором есть функции для быстрого добавления функционала на веб-ресурс. В качестве примера, Google активно применяет язык в ряде собственных проектов, а всё по причине удобной работы и широкого функционала, применимого в любых направлениях программирования. Если метите на место в Гугл, изучение Python – неплохой шанс начать работать в компании.

План курса

В ходе курса вы изучите основы языка Python и научитесь писать консольные программы на нём. Далее вам потребуется изучить дополнительные библиотеки, которые будут расширять и дополнять язык. На нашем сайте вы можете изучить не только Python, но и дополнительные библиотеки. Все дополнительные курсы по Python представлен по этой ссылке.

**#2 - Установка среды разработки**

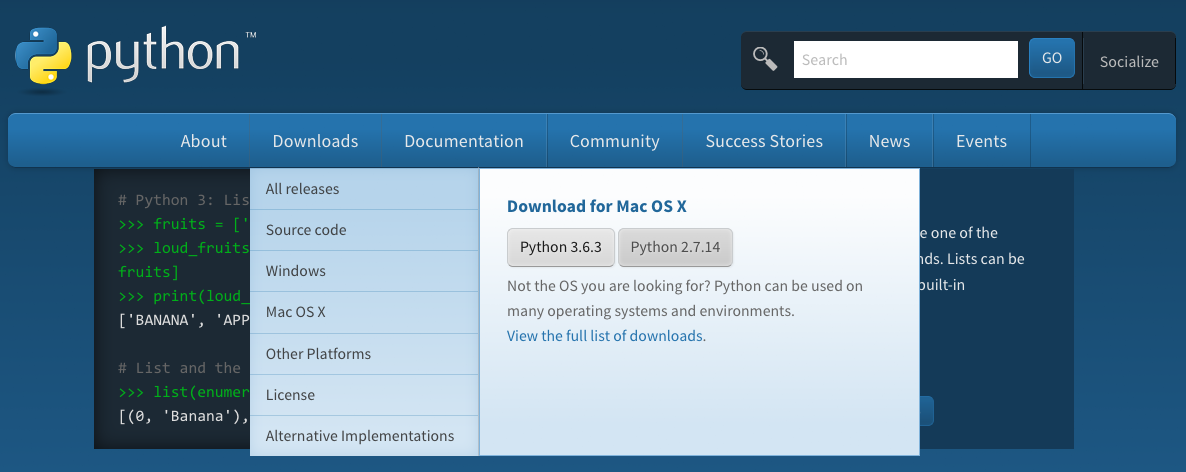
В этом уроке мы с вами установим Python на наш компьютер. Мы установим среду разработки под Python и запустим несколько наших первых файлов, написанных полностью на Python.

1) [Онлайн компилятор](https://www.tutorialspoint.com/execute_python_online.php)  
2)Скачать питон: <https://www.python.org/>

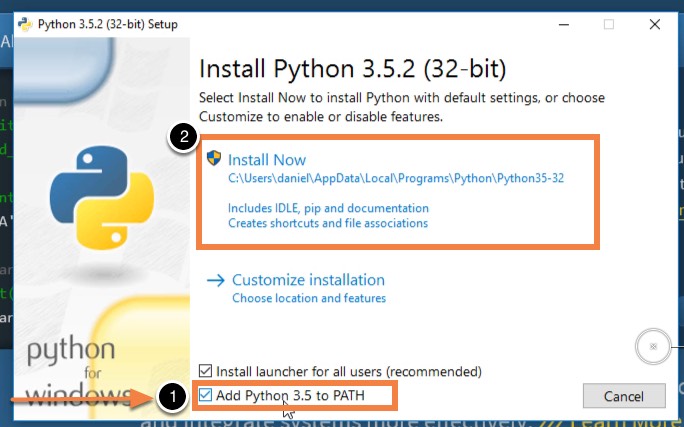
Установка среди разработки это всегда немного сложная тема, так как приходиться все скачивать и настраивать. Поскольку в видео уроке показана информация по поводу установки Python на Маке, то дополнительно здесь мы приведем информацию по поводу установки Python на Windows.

Как установить Python на Windows?

Первым делом вам потребуется скачать сам Питон на компьютер, поэтому перейдите на [официальный сайт](https://python.org/) и скачайте последнюю версию Питона. Скачать можно две разные версии Python 2 и Python 3. В курсе рассматривается новая версия, поэтому спокойно качайте Питон версии 3



После того как все скачается нажмите два раза на установщик и у вас запуститься программа установки Питона. В открывшемся окне установите галочку Add Python X.Y to PATH, затем нажмите на кнопку "Установить" для установки Python.



После установки Python уже должен быть готов к использованию и вам останется лишь протестировать его работоспособность.

Во вкладке "Пуск" должна была появиться новая папка, в которой можно запустить IDLE, если такой папки нет, то попробуйте просто найти эту программу. IDLE это встроенная среда разработки, в которой вы уже можете работать. Также вы можете проверить работоспособность в консоле. Для этого отройте консоль и введите "python3". Если ничего не произошло, тогда попробуйте ввести просто слово "python" и нажать Enter.

Чтобы указать путь к файлу на Windows необходимо использовать следующую конструкцию: C:\Users\USER\Downloads\Python.py. То есть, не надо ставить дополнительные кавычки и прописывать все необходимо через backslash - \.

**#3 - Первая программа (синтаксис)**

В этом уроке мы с вами изучим синтаксис языка Python, а также напишем с вами нашу первую программу на языке Python. Мы разберемся со многими важными моментами, чтобы в дальнейшем к ним не возвращаться!

Иногда у вас могут возникнут ошибки, которые непонятно откуда вообще появились. Чтобы понять как решить какую-либо проблему существует отличная картинка, которая поможет вам во всем разобраться. Картинка ниже:



**#4 - Типы данных и переменные в Python**

В этом уроке мы с вами изучим переменные, а также типы данных в Python. Мы поработаем с переменными, научимся выводить их, присваивать им значения и многое другое.

Переменные невероятно важны, так как позволяют хранить информацию и использовать её в дальнейшем. На старте может быть не понятно зачем что-либо записывать в переменные, если можно оперировать значениями без них. Тем не менее, понимание переменных придет немного позже, когда вы начнёте создавать более сложные программы и вам потребуется хранить информацию в каком-либо месте.

Питон не поддерживает привычное объявление видов переменных, что свойственно для других языков. И всё же они здесь есть. Интерпретатор Python анализирует содержимое переменной и автоматически ставит ей нужный тип.

Во время исполнения программы есть возможность перезаписывать переменные, а также менять их тип. Так, если вначале переменной был задан тип float, то потом её можно преобразовать в строку String.

first\_num = 23.2 # Тип данных float

first\_num = "1" # Тип данных String

При этом объединение нескольких переменных с разными типами данных провоцирует появление ошибки. Что имеется в виду?

first\_num = "IloveYou"

second\_num = 13

res = first\_num + second\_num # Скрипт выдаст ошибку

Всего в Python есть 4 базовых вида переменных:

* some = 1 Integer - целые числа
* some = 1.12 Float - числа с плавающей точкой
* some = "Привет" String - строки
* some = True Boolean - тип данных принимающий либо False, либо True

Есть и другие типы, но их мы будем разбирать в последующих уроках.

Также можно присваивать сразу несколько переменных:

first = sec = third = 1 # Всем трём переменным будет присвоено значение 1

first, sec, third = "Hi", 75, 23.1 # Поочередное присвоение значений

**Исходный код**

num\_1 = float (input ("Enter first num: "))

num\_2 = float (input ("Enter second num: "))

res = float (num\_1) + num\_2

Res = input ("Enter something: ")

Res \*= 5

print (Res)

print ("Result is", res)

**#5 - Условные операторы**

В этом уроке мы с вами рассмотрим условные операторы в языке Python (питон). Благодаря условным операторам if мы можем выполнять разный код в зависимости от различных типов условий

**Что такое условные операторы?**

Они отвечают за изменение поведения программы в зависимости от входных параметров, определённых в проверке. Проще говоря: если будет число 1, то программа запустит скрипт one, а если число 2 – скрипт two. Внутри условных операторов могут быть другие такие же условия для уточнения полученных данных. В рамках одного оператора можно сразу проверить пару условий. Для того, чтобы проверить несколько условий, нужно их разделить элементом and (логическое и).

**Всегда понятнее на примере:**

a = 2

if a != 0 and a != 1:

print ("Проверка сработала")

На экране появляется запись только в том случае, если переменная a не ровняется 0 и 1. То есть обе проверки в операторе должны выдать результат – true.

Есть возможность произвести проверку с помощью or - логическое или. При использовании данного оператора достаточным поводом для запуска сообщения «Заработало» станет соответствие хотя бы одному из условий.

a = 1.1

if a != 1.1 or a > 0:

print ("Проверка сработала")

**Исходный код**

num = input ("Введите число: ")

if int (num) > 0:

if int (num) > 10:

print ("Вы ввели число больше 10")

if int (num) >= 50:

print ("Вы ввели число больше 50")

else:

print ("Вы ввели число меньше 10 и больше 0")

elif int (num) < -10:

print ("Вы ввели число меньше -10")

else:

print ("Вы ввели число меньше 0 и больше -10")

print ("All is okay!")

name = input ()

A = 'Yes' if name != "Test" else 'No'

print (A)

**#6 - Циклы For, While, а также операторы**

В этом уроке мы с вами изучим циклы в языке Python. Благодаря циклам мы можем проделывать операции несколько раз всего за пару строк кода. Мы изучим циклы For, While, а также операторы continue, break и else.

Циклы – основа работы с Питон, они позволяют выполнить одно и то же действие пару раз подряд. Каждый раз могут подставляться разные параметры в цикл. Например, с помощью циклов часто выводят на экран записи с базы данных. Они крайне полезны во время работы с массивами данных. В Python циклы выглядят несколько иначе от своих аналогов в прочих языках.

Практически во всех языках есть цикл do..while, он один раз запускается, а для повторного выполнения требует соответствие каким-то условиям. Здесь подобный цикл отсутствует, но это не значит, что создать подобный алгоритм сложно. Это же можно написать вручную за несколько строк.

Важно! Всегда проверяйте, чтобы в коде не было бесконечных циклов. Результатом их работы становится зависание и прекращение работы программы. Если уж и использовать цикл, то нужно устанавливать интервал обновления в несколько секунд. Ещё можно создать бесконечный цикл и прервать его через оператор break.

a = 10

while True:

a -= 1

if a == 0:

break

В данном примере цикл окажется корректным, ведь break в нужный момент завершит его выполнение.

**Исходный код**

i = 1000

while i > 100:

print (i)

i /= 2

for j in 'hello world':

if j == 'w':

break

else:

print ("Буквы а нету в слове")

**#7 - Списки (list)**

В этом уроке мы с вами изучим списки (list) в языке Python. Списки также часто называют массивами в Питоне. Благодаря спискам мы можем хранить большие данные всего в одной переменной.

Списки можно также назвать массивами, так как выполняют они схожую роль. В Python существует множество разновидностей таких "массивов". В уроке мы рассмотрели лишь первый тип, который называется списком и имеет следующие характеристики:

его элементы можно переопределять;

в него можно добавлять новые элементы;

из него можно удалять элементы;

в качестве ключей идут индексы (числа - 0, 1, 2...).

Это не все характеристики списка, тем не менее это ключевые моменты, которые отличают его от других "массивов" в языке Python.

Для списков существует большое количество функций, которые можно применять для добавления, удаления или же поиска элементов.

В списках можно хранить различную информацию. К примеру, вы можете хранить в нем сведения про всех студентов в группе, например:

lis = ["Андрей", "Иван", "Василий", "Петро", "Максим", "Дима"]

Теперь в одном месте у вас собраны все имена студентов в группе. Согласитесь, это намного удобнее, нежели создавать 10 или даже 20 разных переменных. В дальнейшем мы изучим дополнительные вещи, которые позволят хранить не просто имена студентов, а еще и характеристики каждого из студентов.

l = []

lis = [1, 56, 'x', 34, 2.34, ['S', 't', 'r', 'o', 'k', 'a']]

print (lis)

a = [a + b for a in 'list' if a != 's' for b in 'soup' if b != 'u']

print (a)

l.append (23)

l.append (34)

b = [24, 67]

l.extend (b)

l.insert (1, 56)

l.append (34)

l.remove (34)

l.pop (0)

print (l.index (56))

print (l.count (34))

l.sort ()

l.reverse ()

l.clear ()

print (l)

**#8 - Индексы и срезы**

В этом уроке мы еще поговорим по поводу списков в языке Python. Мы узнаем что такое индексы, а также что такое срезы. Как с ними работать и для чего они вообще нужны!

**Индексы**

Нумерация в списках начинается с нуля, так как список по большей части своей это просто массив, то как в обычном массиве отсчет ведется от 0. Поэтому первый элемент по индексу будет 0, второй - 1, третий - 2 и так далее. Если мы попытаемся взять несуществующий элемент, то это приведет к ошибке.

a = [0, 23, "Hi"] # Список

print (a[4]) # Выдаст ошибку, так как элемента не существует

Очень удобной функцией языка Python является возможность брать элементы с конца при помощи отрицательных индексов. К примеру, если нам нужен второй элемнет с конца, то мы можем записать это так:

a = [0, 23, "Hi", 1.56, 9] # Список

print (a[-2]) # Будет выведено 1.56

**Срезы**

Срезы позволяют обрезать список, взяв лишь те элементы, которые нам будут нужны. Они работают по следующей схеме: list [НАЧАЛО:КОНЕЦ:ШАГ].

* Начало - с какого элемента стоит начать (по умолчанию равна 0);
* Конец - по какой элемент мы берем элементы (по умолчанию равно длине списка);
* Шаг - с каким шагом берем элементы, к примеру каждый 2 или 3 (по умолчанию каждый 1).

Один, несколько или даже все параметры могут быть пропущены.

list[::3] # Берем каждый третий элемент

list[2::2] # Начиная со второго элемента берем каждый второй элемент

list[4:6:] # Начиная с 4 элемента берем все элементы по 6 элемент

list[::] # Берем все элементы

Также могут быть использованы отрицательные числа для срезов.

l = [34, 'sd', 56, 34.34]

i = 0

while i < 4:

print (l[i])

i += 1

print (l[-2::-2])

**#9 - Кортежи (tuple)**

В этом уроке мы с вами научимся работать с кортежами в языке Python. Кортежи это все те же списки, но их нельзя в дальнейшем менять. Их также еще называют tuple.

Кортежи это по сути те же самые списки, только с несколькими отличиями. Кортежи нельзя изменять после их создания, а также они весят чуть меньше, чем списки.

При работе с кортежами действуют абсолютно все методы, о которых было рассказано в уроке про списки. Единственные методы, которые не будут работать, так это методы изменяющие кортеж: удаление, добавление элементов, сложение кортежей и тому подобные.

Кортежи можно создать двумя способами:

* при помощи ключевого слова a = tuple ();
* при помощи литерала a = ().

Их главное преимущество в том, что их нельзя изменять, поэтому ни вы, ни кто-либо другой не сможет как случайно, так и намерено изменить кортеж.

**#10 - Словари (dict), а также их методы**

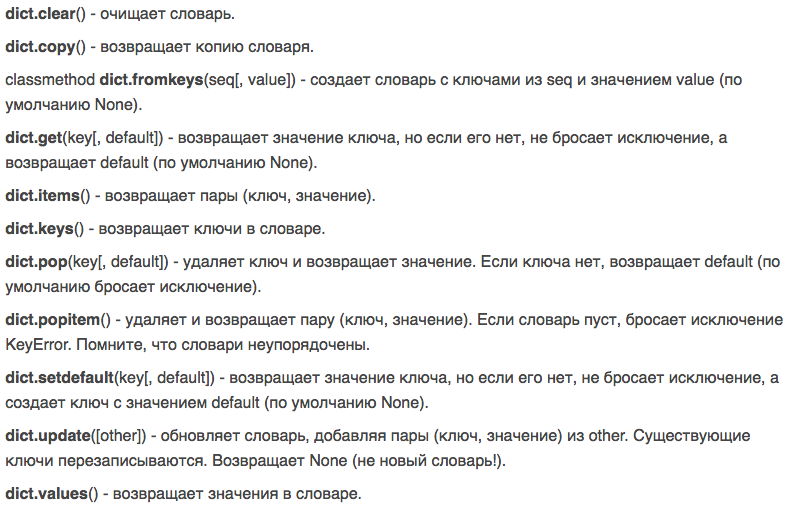
В этом уроке мы с вами рассмотрим работу со словарями в языке Python. Словари очень удобны и работают также как ассоциативные массивы.

Словари отличаются способом записи их, а также индексами. Если в обычном списке в качестве индексов выступают числа (0, 1, 2...), то здесь на их замену приходят "ключи". Так, мы можем создать некий ключ (Студент Федя, например) и добавить к нему целый ряд характеристик (список) в качестве одного элемента. Теперь найти нужного нам студента будет намного проще, так как мы просто будет оперировать ключами, а не числами.

Во многих других языках программирования такие словари зачастую называются ассоциативными массивами, поэтому не редко можно услышать и такое название.

Чтобы создать словарь вы можете воспользоваться следующей конструкцией:

words = {'short': 'Гоша', 'long': 'Георгий'}



d = {a : a \*\* 2 for a in range(7)}

print (d)

person = {'name' : {'last\_name': 'Иванов', 'first\_name': 'Иван', 'middle\_name': 'Иванович'}, 'address': ['г. Андрюшки', 'ул. Васильковская д. 23б', 'кв.12'], 'phone': {'home\_phone': '34-67-12', 'mobile\_phone': '8-564-345-23-65', 'mobile\_phone\_2': 'Нет'}}

# print (person['phone']['mobile\_phone'])

print (person.keys ())

**#11 - Множества (set и frozenset)**

В этом уроке мы с вами изучим множества в языке Python. Множества можно делать set и frozenset. Что это такое, зачем это надо и как с этим всем работать я расскажу в этом уроке!

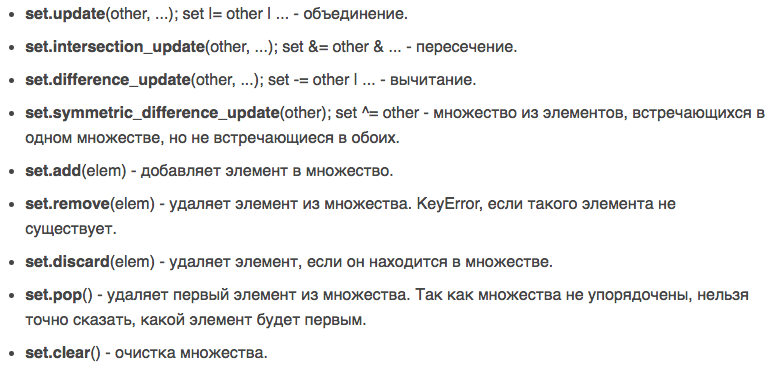
Множества также схожи с массивами, но есть несколько отличий. Во-первых, множества создаются в абсолютно случайном порядке каждый раз. Вы можете разместить элементы как вам будет угодно, но они все равно будут расположены впоследствии в случайном порядке. Во-вторых, множества не могут иметь повторяющихся элементов, поэтому все элементы, которые будут одинаковыми не будут выведены повторно.

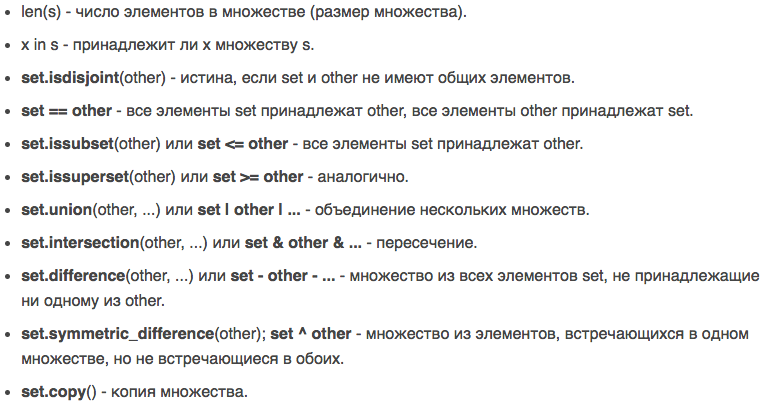
Их очень удобно использовать, если вы хотите удалить повторяющиеся элементы из списка, например

some\_list = [12, 56, 91, 12]

set(some\_list) # Результат: 12, 56, 91

Также для существует огромное множество операций, которые приведены ниже:





**Frozenset**

Frozenset это тоже самое множество, но только которое нельзя в дальнейшем изменять. Можно сказать, что это смесь множества и кортежа.

**#12 - Функции (def, lambda, return)**

В этом уроке мы с вами затронем тему функций в языке Python. Мы разберемся с ключевыми понятиями, такими как: def, lambda и return. Мы научимся создавать функции, вызывать их, а также рассмотрим многие важные моменты.

Функции можно назвать небольшими подпрограммами, куда можно вынести повторяющийся код и обращаться к нему, когда это будет нужно. Функции значительно облегчают построение программ, так как нам не надо копировать однотипный код множество раз, а можно просто воспользоваться одной общей функцией.

В Python функции создаются при помощи ключевого слова def. Каждая функция может иметь какие-либо параметры или же не иметь их вовсе. Также функции способны что-либо возвращать в ходе выполнения кода в них или же ничего не возвращать.

def some\_test(): # Функция без параметров

x = 23

return x # Функция возвращает что-либо

def some\_test\_2(a, b, c): # Функция с 3 параметрами

return a \* b \* c # Функция возвращает результат умножение чисел

def print\_something(word, prefix): # Функция с 2 параметрами

print (prefix, "-", word)

pass # Функция ничего не возвращает

Когда функция ничего не возвращает, то необходимо прописывать ключевое слово pass. Также функции могут возвращать другие функции, тем самым вызывая их. Чтобы обратиться к функции необходимо прописать её названи и передать параметры, если таковы имеются:

res = some\_test\_2 (2, 67, 12)

Здесь мы все записываем в переменную res, так как функция вернет результат, который мы можем куда-то поместить.

**Анонимные функции**

Если функция не должна выполнять большой объем кода, то можно воспользоваться анонимной функцией. Для этого нам потребуется функция lambda. Пример создания такой функции:

multiple = lambda a, b: a \* b

multiple(7, 12)

Такая функция не имеет названия, но её можно присвоить к переменной, которую в дальнейшем необходимо вызывать как обычную функцию.

def func (\*\*args):

return args

print (func (short='dict', longer='dictionary'))

add = lambda x, y: x \* y

print (add (2, 5))

print (add ('q', 5))

print ((lambda x, y: x \* y)(2, 6))

fun = lambda \*args: args

print (fun (2, 56, 78.56))

**#13 - Исключения (Конструкция try - except)**

В этом уроке мы рассмотрим исключения в языке Python. Мы рассмотрим такую конструкцию как try except, а также дополнительно finally и оператор else для исключений.

Исключения это то, что может непредвиденно возникнуть в ходе работы программы. Если такое случиться, то программа просто перестанет работать или же вовсе выключиться. Чтобы такого не допустить вам всегда стоит добавлять исключения при работе с различными вещами. К примеру, вы хотите добавить возможность деления двух чисел? Добавьте также исключение, которое позволит проверять, не является ли делитель нулем. Или же вы хотите что-то вписать файл, но для начала его необходимо открыть. Пропишите исключение, которое будет срабатывать если файл не найден, чтобы программа не пыталась его открывать повторно. Логика я думаю ясна. Чтобы ознакомиться детальнее со всеми возможными исключениями вы можете посетить [эту страницу](https://docs.python.org/3/library/exceptions.html), где приведены все возможные ошибки.

Конструкция самих исключений очень проста. Вам необходимо сначала проверить что-то попробовать сделать, а потом поискать ошибку, если она будет найдена, то выведется ваш код (какая-либо надпись), иначе все сработает корректно и никаких ошибок не будет.

try: # Пробуем что-то сделать

k = 1 / 0 # Деление на ноль

except ArithmeticError: # Отслеживаем на арифметическую ошибку

print ("Найдена ошибка") # Мы нашли ошибку

k = 0 # Наша переменная теперь будет равна 0

print(k) # Просто выводим переменную

**Исходный код**

try:

x = int (input ())

except ValueError:

print ("Вы ввели не число")

x = 0

try:

y = int (input ())

except ValueError:

print ("Вы ввели не число")

y = 0

else:

print ("Все верно")

finally:

print ("Выполнится 100%")

try:

res = x / y

except ZeroDivisionError:

print ("Вы ввели 0")

res = 0

print (res)

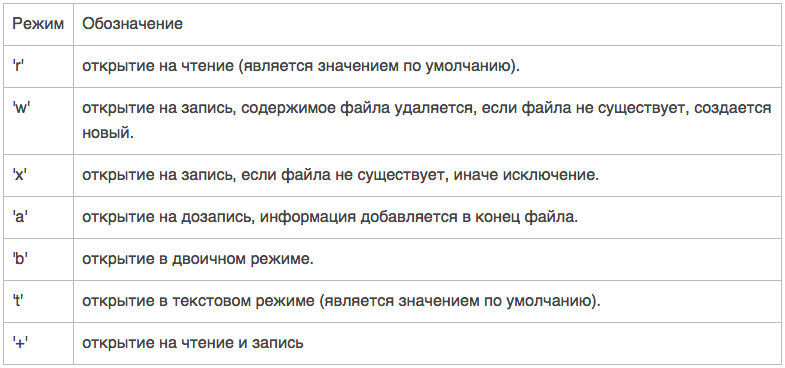
**#14 - Работа с файлами**

В этом уроке мы с вами научимся работать с файлами на языке Python. Мы научимся открывать файлы, записывать в них какую-либо информацию, а также читать всю информацию с какого-либо файла.

Работа с файлами это распространенная вещь в программировании. Нам постоянно что-то надо сохранять или же брать информацию из сохраненных документов.

Перед началом работы с любым текстовым документом важно изначально открыть сам документ. При этом, если его не будет, то это вызовет ошибку. По этой причине, всегда стоит оформлять открытие документов при помощи исключений или же при помощи метода with..as, который будет изучен в следующем уроке.

Для открытия файла существует функция open, которая открывает файл разными способами. Вот все возможные типы открытия файла:



Для записи текста в файл существует метод write, а для чтения метод read. После того, как с файлом была закончена работа его обязательно необходимо закрыть. Это делается при помощи метода close. Рекомендую сразу же при открытии файла его закрывать, чтобы в дальнейшем не забыть это сделать. При этом всю работу с файлом осуществлять между его открытием и закрытием.

**Исходный код**

f = open ('text.txt', 'w')

f.write ('Hi, it's me!

Test')

f.close ()

# for line in f:

# print (line)

**#15 - Менеджеры With ... as**

В этом уроке мы рассмотрим с вами такую конструкцию как With ... as в языке Python. Что она умеет делать, зачем она вообще существует и как с ней работать? Обо всем этом в данном видеоуроке!

With as это конструкция, которая способна выполниться сто процентов. Что это означает? Если при работе с файлами мы используем метод open, то дополнительно нам необходимо еще и закрывать файл, иначе это чревато последствиями.

При работе с With as мы можем быть уверены в том, что файл точно будет закрыт, даже в случае неправильного выполнения функции.

**Исходный код**

with open('test.txt', 'wt', encoding='utf-8') as inFile:

num = int(input())

line = str('1 / ' + str(num) + ' = ' + str(1 / num))

print(line)

inFile.write (line)

**#16 - Модули. Работа с import и from**

В этом уроке мы с вами научимся работать с модулями в языке Python. Мы с вами изучим такие конструкции как import, а также from. Научимся создавать свои собственные модули, а также использовать из стандартной библиотеки Python.

Модули это то, что не просто дополняет язык Python, это делает его полностью другим, добавляя абсолютно новый функционал, который ранее был недоступен. С полным перечнем всех стандартных модулей вы можете ознакомиться [по этой ссылке](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python).

Все модули невозможно запомнить, а главное это не нужно делать. Вам необходимо лишь понимать какой модуль за что отвечает и в случае необходимости вы сможете подсмотреть документацию, которой в интернете полным полно. К примеру, вы хотите создать архив или добавить что-либо в старый? Тогда вам понадобиться модуль ZipFile, который позволяет работать с архивами: создавать их, добавлять файлы, удалять файлы и делать другие вещи.

Таких модулей огромное множество и каждый отвечает за какие-либо функции и новшества.

Чтобы подключить модуль вам необходимо использовать команду import и указать название модуля.

import time # Импорт одного модуля

import random, os # Импорт нескольких модулей

Также вы можете создавать свои модули. Вам просто необходимо создать новый файл и по его названию в дальнейшем вы сможете подключать этот файл как модуль к программе.

Команда From

Чтобы импортировать лишь некоторые методы или же переменные можно использовать команду from. Эта команда записывается следующим образом:

from math import ceil # Импортируем лишь метод ceil

**Псевдонимы для модулей**

Также при работе с модулями можно использовать псевдонимы. Это позволяет сократить название модуля, если оно слишком большое и вам не хочется писать его каждый раз.

import Template as tem # Теперь можно обращаться к модулю как tem

**Исходный код**

import time

import os

import random as r

from module import hi as h, add as a

try:

import nomodule

except ImportError:

print ("Модуля nomodule не существует")

h ()

print (a (45, 15))

def hi ():

print ("Hello World!")

def add (x, y):

return x + y

if \_\_name\_\_ == "\_\_main\_\_":

print ("Hello")

**#17 - Основы ООП Python**

В этом уроке мы с вами приступим к изучению ООП в языке Python. ООП это объектно ориентированное программирование в основе которого является создание класса, а также объекта класса. Мы научимся создавать классы и объекты, а также создадим методы и поля для основного класса!

На начальном этапе ООП – это тёмный лес, в котором многое непонятно и слишком усложнено. Всё совсем не так. Предлагаем абстрагироваться от специфических (непонятных) определений и всё-таки опишем ООП простыми словами. Человек так устроен, что ему становится всё понятнее на примере. Мы возьмём в роли примера робота.

Класс в случае с роботом – это его чертёж. Экземпляром класса называет целый робот, который создан точно по чертежу.  
Наследование – добавление полезных опций к чертежу робота. Мы взяли стандартный чертёж и дорисовали ему лазеры, крылья и броню, потому что у нас заказ на боевую модель.

Полиморфизм – это целая орава роботов, но все они разные, так как созданы на основании различных чертежей. Единственный нюанс – все они берут свои истоки из начального чертежа. Конечная форма наших роботов не имеет принципиального значения. Наша задача сводится к тому, чтобы отдать приказ об уничтожении, а способ достижения цели – это уже не наше дело. Способ зависит от конкретного чертежа, но нас это не касается.

А инкапсуляция является бронёй, защищающей робота. Под пластырем брони находятся уязвимые элементы, вроде проводов и микросхем. После прикрытия брешей с помощью брони (protected или private), робот полностью защищён от внешних вмешательств.

У всех классов методы (функции) могут отличаться, как и поля (переменные) с конструкторами. Каждый класс позволяет создавать любое количество разных объектов, все из них имеют собственные характеристики.  
Чтобы создать класс необходимо использовать ключевое слово class. Далее внутри него можете прописать любые функции и переменные, которые только захотите.

После этого на основе какого-либо класса вы можете создать объект. Для этого используйте название класса для выделения памяти под объект.

class Some:

pass # Класс может ничего не возвращать

obj\_new = Some() # Создание объекта

obj\_second = Some() # Создание 2 объекта

**Исходный код**

class Person:

name = "Ivan"

age = 10

def set(self, name, age):

self.name = name

self.age = age

vlad = Person ()

vlad.set ("Влад", 25)

print (vlad.name + " " + str(vlad.age))

ivan = Person ()

ivan.set ("Иван", 56)

print (ivan.age)

**#18 - Наследование, инкапсуляция, полиморфизм**

В этом уроке мы разберем основные моменты ООП в языке Python, а именно наследование, инкапсуляция, полиморфизм. Эти три понятия являются основой ООП, поэтому их изучить просто необходимо!

Что такое наследование, инкапсуляция и полиморфизм мы уже рассказывали в [предыдущем уроке](https://itproger.com/course/python/17), поэтому особого внимания сейчас на этом концентрировать не будем.

Наследование

Начнем с самого первого, а именно с наследования. Наследование позволяет нам создать второй класс, который будет наследовать все методы, переменные и конструкторы нашего главного класса. Это очень удобно, так как мы можем использовать все необходимые основные методы, но при этом добавить свои собственные, которые будут нужны только лишь в классе наследнике. Для создания наследования используется следующая структура:

class Cars:

wheels = 4 # Общее значение для всех объектов,

# так как все машины имеют колеса

class BMW (Cars):

is\_m\_serias = true # Является ли модель "М" серии?

# Переменная нужна только в классе BMW

Инкапсуляция

Инкапсуляция позволяет нам ограничить доступ к какой-либо функции в классе. Благодаря этому злоумышленники или же мы сами не сможем случайно или намерено вызвать или изменить метод.

class Some:

def \_printWords(self):

print ("Попробуй меня вызвать")

obj = Some()

obj.\_printWords() # Вызов функции ничего не даст

При помощи двойного подчеркивания эффект защиты усиливается, поэтому вызвать функцию вообще будет невозможным.

Полиморфизм

Полиморфизм позволяет нам изменять функции в классах наследниках.

class Cars:

def nothing(self): # Пустая функция

pass

class BMW (Cars):

def nothing(self, word):

print (word + "!") # Функция теперь будет работать по новому

a = BMW()

a.nothing("Some")

**Исходный код**

class Person:

name = "Ivan"

age = 10

def \_\_set(self, name, age):

self.name = name

self.age = age

class Student (Person):

course = 1

igor = Student ()

igor.\_Person\_\_set ("Igor", 19)

igor.course = 2

print (igor.course)

vlad = Person ()

vlad.\_Person\_\_set ("Влад", 25)

print (vlad.name + " " + str(vlad.age))

ivan = Person ()

ivan.\_Person\_\_set ("Иван", 56)

print (ivan.age)

**#19 - Конструкторы, переопределение методов**

В этом уроке мы изучим что такое конструкторы, а также научимся переопределять методы в классах.

Конструкторы позволяют задать некие характеристики для объекта сразу же при его создании. К примеру, у вас есть несколько переменных, которые точно должен иметь объект. Вы можете создать конструктор и указать несколько параметров, которые будут переданы при создании объекта. В одном классе может быть неограниченное количество конструкторов и сам интерпретатор будет понимать к какому конструктору вы обращаетесь. Чтобы создать конструктор необходимо использовать ключевое слово \_\_init\_\_.

class Cars:

wheels = 0

marka = ""

def \_\_init\_\_(self, wheels, marka):

self.wheels = wheels

self.marka = marka

bmw = Cars(4, "X3") # Сразу добавили характеристики

print (bmw.wheels) # Результат - 4

**#20 – Декораторы**

В этом видео мы поговорим о такой функции Python как декораторы. Что это такое, как с ними работать и зачем они нужны? Обо всем этом в данном видеоуроке.

Декораторы бывают очень удобны во многих случаях. Вы можете посчитать, что замечем оборачивать какую-либо функцию с добавлением нового кода до и после её выполнения, если можно просто вписать это же в саму функцию.

Давайте на секунду представим что у нас большая программа и мы используем какую-либо функцию множество раз. Нам не нужно её менять, так как её использует много других подпрограмм. Тем не менее, нам нужно добавить к ней код, который поместить либо в начало функции, либо в её конец. Тут на помощь и приходят декораторы, которые способны выполнить эту задачу.

Одну функцию можно оборачивать несколько раз, как мы это сделали в видео уроке.

**#21 - Заключительный урок**

Это последний видеоурок в видеокурсе по изучению Python. В этом курсе мы изучили с вами немало всего, а теперь вам необходима только практика. Удачи вам в ваших дальнейших проектах!

В этом видеокурсе мы изучили с вами основы языка Python. Чтобы улучшить свои знания и начать создавать новые проекты на этом языке вам необходимо изучить новые библиотеки. Одной из самых популярных библиотек для Python является библиотека [Django](https://www.djangoproject.com/" \t "_blank). У нас на сайте пополняются видеоуроки и следить за всеми курсами, связанными с Python вы можете на [этой странице](https://itproger.com/tag/python).

Также на закрепления знаний мы представляем вам большой видео урок, благодаря которому вы сможете подтянуть знания по курсу и закрепить их. Вы можете посмотреть этот урок ниже или же [перейти на страницу этого урока](https://itproger.com/course/one-lesson/11), чтобы получить дополнительные домашние задания и прочие материалы.