**Table of Content**

**1. Install + First Program**

**2. Variable and Types**

**3. Arithmetic Operators**

**4.** **Python Comparison operators, Logical operators, identity operators, membership operators**

**5. Lists and tuples**

**6. Sets and Dictionaries**

**7. String formatting and methods (print with % and methods i.e. strip, upper, lower, …)**

**8. Conditions (if, elif, else)**

**9. Loops (For loops)**

**10. Loops (while,** **"break" and "continue" statements)**

**11. Functions**

**12. Python Dates**

**13. Classes & Objects**

**14. Exception Handling ( try, log except e as exception)**

**15. Pip install**

**16. Modules and Packages (Importing functions from other py files)**

**1 - Intro Video**

-Hello everyone, and welcome to the very first video in this python for beginners tutorial series.

This series is designed for beginner python programmers with a focus on data science, the aim of this series is establish a solid python programming base from which you will be able to build upon in the future.

-With that said, in this video we will be installing python 3, setting up the system environment variables,

and then creating our very first python program.

-Okay so first things first we need to install python, and we will do that by going to google

typing in download python

-open cmd type python

-open integrated development environment or integrated development and learning environment

-create first program

Print(‘Hello World’) (print create a newline character unlike C).

-test via cmd line

**2. Variable and Types**

welcome to part 2 of this python tutorial series, in this video we are going to be looking at variables and variable types.

To create a variable, you do that by picking a variable name lets pick var1 and then using the equal sign to assign a value to the variable, let’s go with 5

Var1 = 5

Var1

Unlike some other programming languages in python you do not need to declare a variable type when creating the variable.

**Numbers**

-Python supports two types of numbers - integers and floating point numbers.

**Integers**

myint = 7

print(myint)

type(myint)

**Floating point numbers**

myfloat = 7.0

print(myfloat)

myfloat = float(7)

print(myfloat)

type(myfloat)

**Strings**

mystring = 'hello'

print(mystring)

mystring = "hello"

print(mystring)

The difference between the two is that using double quotes makes it easy to include apostrophes (whereas these would terminate the string if using single quotes)

type(mystring)

mystring = "Don't worry about apostrophes"

print(mystring)

Simple operators can be executed on numbers and strings:

one = 1

two = 2

three = one + two

print(three)

hello = "hello"

world = "world"

helloworld = hello + " " + world

print(helloworld)

Assignments can be done on more than one variable "simultaneously" on the same line like this

a, b = 3, 4

print(a,b)

Mixing operators between numbers and strings is not supported:

# This will not work!

one = 1

two = 2

hello = "hello"

print(one + two + hello)

## **Video 3 – Arithmetic Operators**

Addition: +

X= 10, Y=5 z = X+ Y, z

Subtraction: -

X= 10, Y=5 z = X - Y, z

Multiplication: \*

X= 10, Y=5 z = X \* Y, z

Division: /

X= 10, Y=5 z = X / Y, z

Floor Division: //

X= 10, Y=5 z = X // Y, z

Modulus/Remainder: %

Z = 16 % 3, z

Z = 27 % 5, z

Exponentiation: \*\*

Z = 2 \*\* 3, z

## **Video 4 - Python Comparison operators, Logical operators, identity operators, membership operators**

**Comparison Operators**

Equal: ==

Not Equal: !=

Greater than: >

Less than: <

Greater than or equal to: >=

Less than or equal to: <=

**Logical Operators**

And

Or

Not

**Identity Operators**

Is

Is not

**Membership Operators**

In

Not in

## **Video 5 – Lists and tuples,**

List is a collection which is ordered and changeable. Allows duplicate members.

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Welcome to the third video in this python tutorial series. In this video we will be looking at Lists.

**Lists**

Lists are very similar to arrays if you familiar with other languages.

They can contain any type of variable, and they can contain as many variables as you wish.

Mylist = [‘hello’,5,’python’,’java’,6]

mylist = []

mylist.append(1)

mylist.append(2)

mylist.append(3)

print(mylist[0])

print(mylist[1])

print(mylist[2])

for x in mylist:

print(x)

Accessing an index which does not exist generates an exception (an error)

mylist = [1,2,3]

print(mylist[10])

numbers = []

strings = []

names = ["John", "Eric", 5,5,"Jessica"]

**Changing values in a list**

mylist= [1,2,3,4,5,6]

mylist[1] = 333

mylist

**Removing items from a list 3 ways**

Remove, pop, del

remove removes the first matching value, not a specific index:

a = [0, 2, 3, 2]

a.remove(2)

a

del removes the item at a specific index:

a = [3, 2, 2, 1]

del a[1]

a

pop removes the item at a specific index and returns it.

a = [4, 3, 5]

a.pop(1)

**get length of a list**

mylist= [‘soccer’,’hockey’,’tennis’,’basketball’]

len(mylist)

**Tuples**

A tuple is a collection which is ordered and unchangeable. In Python tuples are written with round brackets.

thistuple = ("apple", "banana", "cherry")

print(thistuple)

Once a tuple is created, you cannot change its values, different from a list. Tuples are unchangeable.

thistuple = ("apple", "banana", "cherry")

thistuple[1] = "blackcurrant"

print(thistuple)

Lists in lists

Mylist = [‘DataScience’,5,7.0,[1,2,3,4,5]]

Mylist[3]

Mylist[3][0]

**Video 6 – Sets and Dictionaries**

Dictionary is a collection which is unordered, changeable and indexed. No duplicate members.

**Dictionaries**

A dictionary is a data type similar to arrays, but works with keys and values instead of indexes. Each value stored in a dictionary can be accessed using a key, which is any type of object (a string, a number, a list, etc.) instead of using its index to address it.

phonebook = {}

phonebook["John"] = 938477566

phonebook["Jack"] = 938377264

phonebook["Jill"] = 947662781

print(phonebook)

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

access items or keys

thisdict.items()

thisdict.keys()

get a value by using a key

x = thisdict["model"]

x = thisdict.get("model")

delete an item

del thisdict ["brand"]

print(thisdict)

thisdict.pop("model")

Change an item

thisdict["year"] = 2018

print(thisdict)

Dictionary length

print(len(thisdict))

Clear the dictionary

thisdict.clear()

print(thisdict)

Dictionary Constructor

thisdict = dict(brand="Ford", model="Mustang", year=1964)

print(thisdict)

**Sets**

Set is a collection which is unordered and unindexed. No duplicate members.

thisset = {"apple", "banana", "cherry"}

print(thisset)

add to the set

thisset.add("orange")

Add multiple items to a set, using the update() method:

thisset.update(["orange", "mango", "grapes"])

print(thisset)

Get the number of items in a set:

print(len(thisset))

remove an item from the set

thisset.remove("banana")

thisset.discard("banana")

thisset.pop()

del thisset

print(thisset)

Clear the entire set

thisset.clear()

print(thisset)

The set() Constructor

thisset = set(("apple", "banana", "cherry"))

print(thisset)

Set intersections (Return a set that contains the items that exist in both set x, and set y)

x = {"apple", "banana", "cherry"}

y = {"google", "microsoft", "apple"}

z = x.intersection(y)

print(z)

set unions

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.union(y)   
  
print(z)

**Video 7 – String formatting and basic string methods**

Python uses C-style string formatting to create new, formatted strings

name = "John"

print(“Hello, John!”)

print(“Hello, “ + name + “!”)

print("Hello, %s!" % name)

To use two or more argument specifiers, use a tuple (parentheses):

name = "John"

age = 23

print("%s is %d years old." % (name, age))

**String Methods**

**Strip**

a = " Hello, World!

print(a.strip())

**Length**

a = "Hello, World!"  
print(len(a))

Lower

a = "Hello, World!"  
print(a.lower())

upper

a = "Hello, World!"  
print(a.upper())

a = "Hello, World!"  
print(a.replace("H", "J"))

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

**Video 8 - Conditions (if, elif, else)**

indentation is important here

**If**

a = 33  
b = 200

if b > a:

print("b is greater than a")

**ElIF**

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")

**Else**

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a"

if a > b and c > a:  
  print("Both conditions are True")

if a > b or a > c:  
  print("At least one of the conditions is True")

**Video 9 – For Loop**

**# Prints out the numbers 0,1,2,3,4**

**for x in range(5):**

**print(x)**

**# Prints out 3,4,5**

**for x in range(3, 6):**

**print(x)**

**# Prints out 3,5,7**

**for x in range(3, 8, 2):**

**print(x)**

For loops iterate over a given sequence. Here is an example:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)

Loop through the letters in the word "banana":

for x in "banana":  
  print(x)

The break Statement

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)   
  if x == "banana":  
    break

Do not print banana:

fruits = ["apple", "banana", "cherry"]

for x in fruits:

if x == "banana":

continue

print(x)

Nested Loops

adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
  for y in fruits:  
    print(x, y)

**Video 10 – While Loop**

**With the while loop we can execute a set of statements as long as a condition is true.**

i = 1  
while i < 6:  
  print(i)  
  i += 1

**Exit the loop when i is 3:**

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

Continue to the next iteration if i is 3:

i = 0  
while i < 6:  
  i += 1   
  if i == 3:  
    continue  
  print(i)

**Video 11 - Functions**

**What are Functions?**

**Functions are a convenient way to divide your code into useful blocks, allowing us to order our code, make it more readable, reuse it and save some time. Also functions are a key way to define interfaces so programmers can share their code.**

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

Functions may also receive arguments (variables passed from the caller to the function). For example:

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

**def my\_function\_with\_args(username, greeting):**

**print("Hello, %s , From My Function!, I wish you %s"%(username, greeting))**

Functions may return a value to the caller, using the keyword- 'return' . For example:

**def sum\_two\_numbers(a, b):**

**return a + b**

**Video 12 - Python Dates**

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

import datetime  
  
x = datetime.datetime.now()  
print(x)

When we execute the code from the example above the result will be:

2019-01-21 15:34:14.718088

The date contains year, month, day, hour, minute, second, and microsecond.

The datetime module has many methods to return information about the date object.

Here are a few examples, you will learn more about them later in this chapter:

import datetime  
  
x = datetime.datetime.now()  
  
print(x.year)  
print(x.strftime("%A"))

## **Creating Date Objects**

To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

import datetime  
  
x = datetime.datetime(2020, 5, 17)  
  
print(x)

## **The strftime() Method**

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

import datetime  
  
x = datetime.datetime(2018, 6, 1)  
  
print(x.strftime("%B"))

**Video 13 - Classes & Objects**

**Python Classes/Objects**

**Python is an object oriented programming language.**

**Almost everything in Python is an object, with its properties and methods.**

**A Class is like an object constructor, or a "blueprint" for creating objects.**

**Create a Class**

Create a class named MyClass, with a property named x:

class MyClass:  
  x = 5

## **Create Object**

Now we can use the class named myClass to create objects:

Create an object named p1, and print the value of x:

p1 = MyClass()  
print(p1.x)

## **The \_\_init\_\_() Function**

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in \_\_init\_\_() function.

All classes have a function called \_\_init\_\_(), which is always executed when the class is being initiated.

Use the \_\_init\_\_() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Create a class named Person, use the \_\_init\_\_() function to assign values for name and age:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)

## **Object Methods**

Objects can also contain methods. Methods in objects are functions that belongs to the object.

Let us create a method in the Person class:

Insert a function that prints a greeting, and execute it on the p1 object:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

## **The self Parameter**

The self parameter is a reference to the class itself, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Use the words *mysillyobject* and *abc* instead of *self*:

class Person:  
  def \_\_init\_\_(mysillyobject, name, age):  
    mysillyobject.name = name  
    mysillyobject.age = age  
  
  def myfunc(abc):  
    print("Hello my name is " + abc.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

## **Modify Object Properties**

You can modify properties on objects like this:

Set the age of p1 to 40:

p1.age = 40

## **Delete Object Properties**

You can delete properties on objects by using the del keyword:

Delete the age property from the p1 object:

del p1.age

## **Delete Objects**

You can delete objects by using the del keyword:

Delete the p1 object:

del p1

**Video 14: Exception Handling (try, log except e as exception)**

The try block lets you test a block of code for errors.

The except block lets you handle the error.

The finally block lets you execute code, regardless of the result of the try- and except blocks.

## **Exception Handling**

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the try statement:

The try block will generate an exception, because x is not defined:

try:  
  print(x)  
except:  
  print("An exception occurred")

## **Many Exceptions**

You can define as many exception blocks as you want, e.g. if you want to execute a special block of code for a special kind of error:

Print one message if the try block raises a NameError and another for other errors:

try:  
  print(x)  
except NameError:  
  print("Variable x is not defined")  
except:  
  print("Something else went wrong")

## **Else**

You can use the else keyword to define a block of code to be executed if no errors were raised:

In this example, the try block does not generate any error:

try:  
  print("Hello")  
except:  
  print("Something went wrong")  
else:  
  print("Nothing went wrong")

## **Finally**

The finally block, if specified, will be executed regardless if the try block raises an error or not.

try:  
  print(x)  
except:  
  print("Something went wrong")  
finally:  
  print("The 'try except' is finished")

Printing The exception

try:

print(x)

except Exception as e:

print(e)

**Logging the exception**

import datetime

try:

print(x)

except Exception as e:

today = datetime.datetime.today().strftime('%Y-%m-%d')

f = open("C:\\Users\\amghader\\Desktop\\demofile.txt",'a')

f.write(str(e)+ " | " + today + "\n" )

f.close()

**Video 15 - Pip (package manager)**

PIP is a package manager for Python packages, or modules if you like.

A package contains all the files you need for a module.

Modules are Python code libraries you can include in your project.

## **Check if PIP is Installed**

Navigate your command line to the location of Python's script directory, and type the following:

### **Example**

Check PIP version:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip --version

## **Download a Package**

Downloading a package is very easy.

Open the command line interface and tell PIP to download the package you want.

Navigate your command line to the location of Python's script directory, and type the following:

### **Example**

Download a package named "camelcase":

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip install camelcase

## **Using a Package**

Once the package is installed, it is ready to use.

Import the "camelcase" package into your project.

### **Example**

Import and use "camelcase":

import camelcase  
  
c = camelcase.CamelCase()  
  
txt = "hello world"  
  
print(c.hump(txt))

## **List Packages**

Use the list command to list all the packages installed on your system:

### **Example**

List installed packages:

C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>pip list

Show how to install packages from the Python Package Index!

**Video 16 - Modules and Packages (Importing functions from other py files)**

## **What is a Module?**

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

## **Create a Module**

To create a module just save the code you want in a file with the file extension .py:

### **Example**

Save this code in a file named mymodule.py

def greeting(name):  
  print("Hello, " + name)

## **Use a Module**

Now we can use the module we just created, by using the import statement:

### **Example**

Import the module named mymodule, and call the greeting function:

import mymodule  
  
mymodule.greeting("Jonathan")

## **Variables in Module**

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

### **Example**

Save this code in the file mymodule.py

person1 = {  
  "name": "John",  
  "age": 36,  
  "country": "Norway"  
}

### **Example**

Import the module named mymodule, and access the person1 dictionary:

import mymodule  
  
a = mymodule.person1["age"]  
print(a)

Import only the person1 dictionary from the module:

from mymodule import person1  
  
print (person1["age"])