Learn Python in one Video

Python Programming Tutorial

How to install python?

It is so easy on both mac and pc.
For Mac: Go to Python.org/download. And specifically, go to Python 3.4.2 and click on that to see all the different ways you can install it. Choose Mac OS. Continue and agree and install. After installation goes into my application go to python 3.4 either use IDLE or pycharm.

For window: Go to Python.org/download. Choose windows X86 MSI installer to click run. Click next and choose a directory. Continue and finish, by clicking windows you can see python idle.

PyCharm:
To get community edition pycharm go to jetbrains.com/pycharm/downloads and click on the download after selecting the operating system you are using. Select and install.

Open pycharm. Go to preferences. Make sure you have a right interpreter. Click on python console to check the python version is same as what is just installed. Go to project interpreter to check if the python 3.4 is selected.

Create a new project:
Name it as hello.py. Py is the extension used for python applications
Import the modules which are going to provide us with some special functions that we are going to use, such as a random generator, sys module, and the operating system module(OS).

```
# import is used to make specialty functions available
# These are called modules
import random
import sys
import os
```

Simple hello world:

To do a simple hello world print, use single or double quotes inside the print.

```
# Hello world is just one line of code
# print() outputs data to the screen
print("Hello World")
```

Run it. To see hello world printed.
To comment a single line # symbol can be used.
# Comment
To comment multi-line use ‘’’ (three single quotes)

```
'''
This is a multi-line comment
'''
```

Like this, we can handle comments inside python.

Variables:

Variable is just a place we are going to store values and we can store pretty much anything in python.

```
# A variable is a place to store values
# Its name is like a label for that value
name = "its a string"
print(name)
```

The value can be in between single or double quotes
A variable should start with a letter and then it can have numbers or it can have underscores.

Data types:

There are five main data types in python. Numbers, Strings, lists, tuples and dictionaries or maps.

1. **Numbers:**

There are seven different different arithmetic operators. +, -, *, /, %(modulus), **(exponent) and // (floor division - perform division and discard remainder)

Examples:

```
print("9 + 2 =", 9+2)
print("9 - 2 =", 9-2)
print("9 * 2 =", 9*2)
print("9 / 2 =", 9/2)
print("9 % 2 =", 9%2)
print("9 ** 2 =", 9**2)
print("9 // 2 =", 9//2)
```

Output print:

```
9 + 2 = 11
9 - 2 = 7
9 * 2 = 18
9 / 2 = 4.5
9 % 2 = 1
9 ** 2 = 81
9 // 2 = 4
```

When you are doing the arithmetic operation, understand the order of operation matters. If we have multiplication or division they are going to be performed before addition or subtraction.

```
print("1 + 2 - 3 * 2 =", 1 + 2 - 3 * 2)
```

Here we are adding 1 plus 2 minus 3 times 2

```
print("(1 + 2 - 3) * 2 =", (1 + 2 - 3) * 2)
```

Here we are adding 1 plus 2 minus 3 and the values is multiplied by 2.

If you want to perform some arithmetic, just put a comma right after that string. The order of operation definitely matters, use braces anytime you can when working with arithmetic operations.

2. **Strings:**

A string i going to be a string of characters, numbers which are between either single or double quote. If you want to include a quote inside of it, just put a backslash and then follow that with a double quote. And in python, there is no semicolon at the end of the line like most programming languages.

```
quote = "\"Always remember your unique,"
print(quote)
"Always remember your unique,
```

For a multi-line string use three single quotes like

```
multi_line_quote = ''' just
like everyone else" '''
print(multi_line_quote)
```

Output:

```
 just
like everyone else"
```

To join two strings, use + between the strings like `print(quote + multi_line_quote)`
Which gives a print output as

```
"Always remember your unique, just
like everyone else"
```

Different formats to use print:
%s is used to embed a string in output

```
print("%s %s %s" % ('I like the quote', quote, multi_line_quote))
```

Output:

```
I like the quote "Always remember your unique,  just
like everyone else"
```

To avoid printing new lines every time end ="" is used.

```
print("I don't like ",end="")
print("newlines")
print("ok")
```

Output:
I don't like newlines
ok

```
# You can print a string multiple times with *
print('\n' * 5) # prints 5 new lines

print('two times three ' * 2)
Output:
two times three two times three
```

3. Lists:

Lists allow you to create a list of values and manipulate them. Each value will have an index with the first one having the index of 0 and the index is just like a label.

Creating a list:
Use a square bracket and put everything you want to store inside. The values in the [] can be numbers, strings, and other data types.

```
# A list allows you to create a list of values and manipulate them
# Each value has an index with the first one starting at 0
# Print a first item of the list

grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']
print('The first item is', grocery_list[0])

Output:
The first item is Juice
```

You can change the value stored in a list as below.

```
grocery_list[0] = "No Juice"
print(grocery_list)

Output:
['No Juice', 'Tomatoes', 'Potatoes', 'Bananas']
```

To print a subset of a list:
Print the grocery list with starting index from 1 and print up to index three and not including index three.

```
# You can get a subset of the list with [min: up to but not including
max]

print(grocery_list[1:3])

Output:
['Tomatoes', 'Potatoes']

# You can put any data type in a list including a list
other_events = ['Wash Car', 'Pick up Kids', 'Cash Check']
```

Stores those in a string by putting an equal sign with a string name string variable on the side. We can put a list of the list inside of a list.

```
to_do_list = [other_events, grocery_list]

print(to_do_list)
Output:
[['Wash Car', 'Pick up Kids', 'Cash Check'], ['No Juice', 'Tomatoes',
'Potatoes', 'Bananas']]
```

If we want the second item out of the second list, we can give 1 and 1 to get a second list and the second item on that list.

```
# Get the second item in the second list (Boxes inside of boxes)
print(to_do_list[1][1])

Tomatoes
```

Append items:
Can be done by using keyword append. It appends to the information which is already there.

```
# You add values using append
grocery_list.append('onions')
print(to_do_list)

Output:
[['Wash Car', 'Pick up Kids', 'Cash Check'], ['No Juice', 'Tomatoes',
'Potatoes', 'Bananas', 'onions']]
```

Insert to a specific position:
1 represents to insert in the second position.

```
# Insert item at given index
grocery_list.insert(1, "Pickle")
print(grocery_list)
Output:
['No Juice', 'Pickle', 'Tomatoes', 'Potatoes', 'Bananas', 'onions']

# Remove item from list
grocery_list.remove("Potatoes")
print(grocery_list)
Output:
['No Juice', 'Pickle', 'Tomatoes', 'Bananas', 'onions']

# Sorts items in list
grocery_list.sort()
print(grocery_list)
Output:
['Bananas', 'No Juice', 'Pickle', 'Tomatoes', 'onions']

# Reverse sort items in list
grocery_list.reverse()
print(grocery_list)
Output:
['onions', 'Tomatoes', 'Pickle', 'No Juice', 'Bananas']

# del deletes an item at specified index
del grocery_list[4]
print(to_do_list)
Output:
[['Wash Car', 'Pick up Kids', 'Cash Check'], ['onions', 'Tomatoes',
'Pickle', 'No Juice']]
```

We can combine strings inside of list and we can also combine lists together.

```
# We can combine lists with a +
to_do_list2 = other_events + grocery_list
print(to_do_list2)

Output:
['Wash  Car',  'Pick  up  Kids',  'Cash  Check',  'onions',  'Tomatoes',
'Pickle', 'No Juice']
```

Get the length of the list to the number of items of to do list by using len() function. We could get the maximum string by calling a max function which returns whatever is highest alphabetically. We can also get the minimum item like whatever comes first alphabetically using min function.

```
# Get length of list
print(len(to_do_list2))
7

# Get the max item in list
print(max(to_do_list2))
onions

# Get the minimum item in list
print(min(to_do_list2))
Cash Check
```

4. **Tuples:**

Tuples are similar to a list. But tuples is that unlike lists we are not going to be able to change a tuple after it is created. Use () for the tuple.
Create a tuple called as pi_tuple -

```
# Values in a tuple can't change like lists

pi_tuple = (3, 1, 4, 1, 5, 9)
```

Why tuples are useful is sometimes you have data that you do not want easily to be changed. If you want to change a tuple you can convert it into a list and then change it there and convert it back to a tuple but that’s not normally done.

```
# Convert tuple into a list
new_list = list(pi_tuple)
print(new_list)

[3, 1, 4, 1, 5, 9]

new_list.remove(1)
print(new_list)

[3, 4, 1, 5, 9]

# Convert a list into a tuple
new_list_tuple = tuple(new_list)
```

```
print(new_list_tuple)

(3, 4, 1, 5, 9)
```

You can also get the length of the tuple by just using len and then the tuple. Min and Max are going to work in exactly the same way as lists.

```
# tuples also have len(tuple), min(tuple) and max(tuple)
print(len(new_list_tuple))
5
print(max(new_list_tuple))
9
print(min(new_list_tuple))
1
```

There are the different functions you are going to use with tuples

5. Dictionaries/ Maps:

Dictionary is going to be made up of values with a unique key for each value you are going to be storing and they are very similar to lists but you can't join dictionaries together like you can with lists with a plus sign. That's the main difference between lists and dictionaries.

Create a dictionary:
Use {} like curly braces in a dictionary. In braces type the key followed by a colon and then whatever the value would be.

```
# Made up of values with a unique key for each value
# Similar to lists, but you can't join dicts with a +

super_villains = {'Fiddler' : 'Isaac Bowin',
                  'Captain Cold' : 'Leonard Snart',
                  'Weather Wizard' : 'Mark Mardon',
                  'Mirror Master' : 'Sam Scudder',
                  'Pied Piper' : 'Thomas Peterson'}
```

To find the corresponding value for the key, use square brackets and type in the key.

```
print(super_villains['Captain Cold'])

Leonard Snart
```

Delete, replace and len functions are similar to lists.

```
# Delete an entry
del super_villains['Fiddler']
print(super_villains)

{'Pied Piper': 'Thomas Peterson', 'Mirror Master': 'Sam Scudder', 'Weather Wizard': 'Mark Mardon', 'Captain Cold': 'Leonard Snart'}

# Replace a value
super_villains['Pied Piper'] = 'Hartley Rathaway'
```

```
# Print the number of items in the dictionary
print(len(super_villains))
4

# Get the value for the passed key
print(super_villains.get("Pied Piper"))
Hartley Rathaway

# Get a list of dictionary keys
print(super_villains.keys())
dict_keys(['Pied Piper', 'Mirror Master', 'Weather Wizard', 'Captain
Cold'])

# Get a list of dictionary values
print(super_villains.values())
dict_values(['Hartley Rathaway', 'Sam Scudder', 'Mark Mardon', 'Leonard
Snart'])
```

Conditionals:

The statements if-else and elif are used to form different actions based off of different conditions and those conditions are going to be whether two things are going to be equal, not equal, greater than, greater than or equal to, less than, less than or equal to.

If/ if -else/ if-elif-else:
If the statement is going to execute code if a condition is met and as you are going to see here white space is used to group blocks of code inside of python in model different ways.

Make sure your whitespaces are always the same but if you are using pycharm or some other IDE its more than likely is going to do that for you.

```
# The if, else and elif statements are used to perform different
# actions based off of conditions
# Comparison Operators : ==, !=, >, <, >=, <=

# The if statement will execute code if a condition is met
# Whitespace is used to group blocks of code in Python
# Use the same number of preceding spaces for blocks of code

age = 30
if age > 16 :
   print('You are old enough to drive')

You are old enough to drive

# Use an if statement if you want to execute different code regardless
# of whether the condition is met or not

if age > 16 :
   print('You are old enough to drive')
```

```
else :
   print('You are not old enough to drive')
```

```
You are old enough to drive
```

If you want to check for multiple conditions this is where elif pops in.

```
# If you want to check for multiple conditions use elif
# If the first matches it won't check other conditions that follow

if age >= 21 :
   print('You are old enough to drive a tractor trailer')
elif age >= 16:
   print('You are old enough to drive a car')
else :
   print('You are not old enough to drive')
```

```
You are old enough to drive a tractor trailer
```

The single quotes and double quotes they don’t really matter in print above.

We can combine conditions with what are called logical operators. Logical operators are and, or, not. They provide us with a way to have multiple conditions match. It's kind of explanatory the way they work. Once your condition is matched we are no longer going to check any other conditions below.

```
# You can combine conditions with logical operators
# Logical Operators: and, or, not

if ((age >= 1) and (age <= 18)):
   print("You get a birthday party")
elif (age == 21) or (age >= 65):
   print("You get a birthday party")
elif not(age == 30):
   print("You don't get a birthday party")
else:
   print("You get a birthday party yeah")
```

```
You get a birthday party yeah
```

Loops:

Looping is just going to allow us to perform an action a set number of times. One way of doing that is with for loop and there are other different ways too.

For:
If you want to perform a task suppose for 10 times. We are going to start at 0 and till 10 but it doesn’t go till 10 and ends up at 9 using range(0,10).

```
# Allows you to perform an action a set number of times
# Range performs the action 10 times 0 - 9
```

```
for x in range(0, 10):
   print(x , ' ', end="")

print('\n')
0  1  2  3  4  5  6  7  8  9


# You can use for loops to cycle through a list
grocery_list = ['Juice', 'Tomatoes', 'Potatoes', 'Bananas']

for y in grocery_list:
   print(y)
Juice
Tomatoes
Potatoes
Bananas

# You can also define a list of numbers to cycle through
for x in [2,4,6,8,10]:
   print(x)
2
4
6
8
10
```

We can double loops to cycle loop list inside a list.

```
# You can double up for loops to cycle through lists
num_list =[[1,2,3],[10,20,30],[100,200,300]];

for x in range(0,3):
   for y in range(0,3):
       print(num_list[x][y])

1
2
3
10
20
30
100
200
300
```

While:
Take a look at another way of working with looping. That's the while loop inside python. While loops are basically to be used when you have no idea ahead of time how many times you are going to need to loop.

```
# While loops are used when you don't know ahead of time how many
# times you'll have to loop
# Random generator is used to get something random
# Cycles through the whole bunch of random numbers until 2 is generated

random_num = random.randrange(0,5)

while (random_num != 2):
   print(random_num)
   random_num = random.randrange(0,5)
1
1
4
3
0
3
```

Using while loop in another traditional manner as like a for a loop.

```
# An iterator for a while loop is defined before the loop
i = 0;
while (i <= 20):
   if(i%2 == 0):
       print(i)
   elif(i == 9):
       # Forces the loop to end all together
       break
   else:
       # Shorthand for i = i + 1
       i += 1
       # Skips to the next iteration of the loop
       continue

   i += 1
0
2
4
6
8
```

In += symbol + can be replaced by -,*,/.

The break is used to force ourselves out of the loop completely and not continue checking. Continue is used to skip the rest of the stuff that comes that's inside of the while loop that comes after this, it's going to jump back up to the while loop.

Functions:

Functions which are going to allow us to both reuse and write more readable code. And how you define a function is you type in def (define) and the function name and followed by the parameter that it is going to receive. To return something to the caller of the function just type in return. Make sure you define the function before you call for them to use. To call a function, use the function name and pass the parameters. Another thing that is important to know is the variables which are created inside of the function are not available outside of the function because it doesn’t exist outside.

```
# Functions allow you to reuse and write readable code
# Type def (define), function name and parameters it receives
# return is used to return something to the caller of the function
def addNumbers(fNum, sNum):
   sumNum = fNum + sNum
   return sumNum

print(addNumbers(1, 4))
5

# Can't get the value of sumNum because it was created in a function
# It is said to be out of scope
# print(sumNum)

# If you define a variable outside of the function it works every place
not the value
newNum = 0;
def subNumbers(fNum, sNum):
   newNum = fNum - sNum
   return newNum

print(subNumbers(1, 4))
-3
print(newNum)
0
```

Taking input from User:

Another thing that is useful is to be able to get input from the user. To get the value users type in you can use sys.stdin.readline()

```
print('What is your name?')

# Stores everything typed up until ENTER
name = sys.stdin.readline()

print('Hello', name)

What is your name?
>? vaishnavi
Hello vaishnavi
```

More about strings…

```
# A string is a series of characters surrounded by ' or "
```

```
long_string = "I'll catch you if you fall - The Floor!?"

# Retrieve the first 4 characters
print(long_string[0:4])

I'll

# Get the last 5 characters
print(long_string[-7:])

Floor!?

# Everything up to the last 5 characters
print(long_string[:-10])

I'll catch you if you fall - T

# Concatenate part of a string to another
print(long_string[:4] + " be there")

I'll be there

# String formatting
# %c for character, %s for string, %d for integer, %.5f upto 5 decimal
places
print("%c is my %s letter and my number %d number is %.5f" % ('X',
'favorite', 1, .14))

X is my favorite letter and my number 1 number is 0.14000

# Capitalizes the first letter
print(long_string.capitalize())
I'll catch you if you fall - the floor!?

# Returns the index of the start of the string
# case sensitive
print(long_string.find("Floor"))
33

# Returns true if all characters are letters ' isn't a letter
print(long_string.isalpha())
False

# Returns true if all characters are numbers
print(long_string.isalnum())
False

# Returns the string length
print(len(long_string))
40
```

```
# Replace the first word with the second (Add a number to replace more)
print(long_string.replace("Floor", "Ground"))
I'll catch you if you fall - The Ground!?

# Remove white space from front and end
print(long_string.strip())
I'll catch you if you fall - The Floor!?

# Split a string into a list based on the delimiter you provide
quote_list = long_string.split(" ")
print(quote_list)
["I'll", 'catch', 'you', 'if', 'you', 'fall', '-', 'The', 'Floor!?']
```

File i/o or Input/Output:

If you want to create as well as open a file using open followed by file name and the mode.

```
# Overwrite or create a file for writing
# Use ab+ to read and append to file
test_file = open("testfile.txt", "wb")

# Get the file mode used
print(test_file.mode)
wb

# Get the files name
print(test_file.name)
testfile.txt

# Write text to a file with a newline
test_file.write(bytes("Write me to the file\n", 'UTF-8'))

# Close the file
test_file.close()

# Opens a file for reading and writing
test_file = open("testfile.txt", "r+")

# Read text from the file
text_in_file = test_file.read()

print(text_in_file)
 Write me to the file
test_file.close()
# Delete the file
# Import os before using this
os.remove("testfile.txt")
```

Remember to close the file after working on it.

Classes and Objects:

The concept of object-oriented programming allows us to model real-world things using code. So just like real-world object has attributes like color, height, weight and every real-world object has abilities like walk, talk and eat we are going to define attribute using variables inside of things called classes and the abilities are going to be the functions.

Now we define these real-world objects by using a class which is a blueprint for creating animals here. None can be used to signify the lack of a value for an attribute. By preceding these variables with two underscores, they are going to be private. That means if you want to change the values of the variables, we are going to need to use a function inside of the class that allows to change it. And to get those values we are also going to use a function inside of the class.

The word 'self' means it allows the object to refer itself inside of the class. Self-works like this in other languages too. The code is called encapsulation (verifies whether data is valid and good). A constructor is going to be called to set up or initialize an object inside a class. Define and the constructor is two underscores and init and two more underscores. Whenever the object is created we demand that all the values are passed in and then we initialize. Create setters and getters for all the different variables.

```
# The concept of OOP allows us to model real-world things using code
# Every object has attributes (color, height, weight) which are object
variables
# Every object has abilities (walk, talk, eat) which are object functions

class Animal:
   # None signifies the lack of a value
   # You can make a variable private by starting it with __
   __name = None
   __height = None
   __weight = None
   __sound = None

   # The constructor is called to set up or initialize an object
   # self allows an object to refer to itself inside of the class
   def __init__(self, name, height, weight, sound):
       self.__name = name
       self.__height = height
       self.__weight = weight
       self.__sound = sound

   def set_name(self, name):
       self.__name = name

   def set_height(self, height):
       self.__height = height

   def set_weight(self, height):
       self.__height = height
```

```python
    def set_sound(self, sound):
        self.__sound = sound

    def get_name(self):
        return self.__name

    def get_height(self):
        return str(self.__height)

    def get_weight(self):
        return str(self.__weight)

    def get_sound(self):
        return self.__sound

    def get_type(self):
        print("Animal")

    def toString(self):
    return   "{}   is   {}   cm   tall   and   {}   kilograms   and   says
{}".format(self.__name, self.__height, self.__weight, self.__sound)

# How to create a Animal object
cat = Animal('Whiskers', 33, 10, 'Meow')
print(cat.toString())

# You can't access this value directly because it is private
#print(cat.__name)
```

Inheritance means whenever you inherit from another class that you automatically get all the variables and methods or functions that are created in the class you are inheriting from. Here Dog is inherited from the Animal class and with a redefined constructor.

```python
# You can inherit all of the variables and methods from another class
class Dog(Animal):
    __owner = None

    def __init__(self, name, height, weight, sound, owner):
        self.__owner = owner
        self.__animal_type = None

        # How to call the superclass constructor
        super(Dog, self).__init__(name, height, weight, sound)

    def set_owner(self, owner):
        self.__owner = owner

    def get_owner(self):
        return self.__owner
    def get_type(self):
        print ("Dog")

    # We can overwrite functions in the super class
    def toString(self):
        return "{} is {} cm tall and {} kilograms and says {}. " \
               "His       owner       is       {}".format(self.get_name(), 
self.get_height(), 
                                                  self.get_weight(), 
self.get_sound(), self.__owner)
```

Method overloading means that that you are able to perform different tasks based off of the attributes that are sent in.

```
    # You don't have to require attributes to be sent
    # This allows for method overloading
    def multiple_sounds(self, how_many=None):
        if how_many is None:
            print(self.get_sound())
        else:
            print(self.get_sound() * how_many)

spot = Dog("Spot", 53, 27, "Ruff", "Vaishnavi")
print(spot.toString())
```

Polymorphism sounds really complicated, but it basically just allows you to refer to objects as their superclass and then have the correct functions called automatically.

```
# Polymorphism allows us to refer to objects as their superclass
# and the correct functions are called automatically
class AnimalTesting:
    def get_type(self, animal):
        animal.get_type()

test_animals = AnimalTesting()

test_animals.get_type(cat)
test_animals.get_type(spot)

spot.multiple_sounds(4)
spot.multiple_sounds()
```

Code:

➔ https://github.com/rvaishnavi/lpiov

Reference:

➔ https://www.youtube.com/watch?v=N4mEzFDjqtA&feature=youtu.be by Derek Banas