Topics in Python need to be discussed:

1. How to include multiple statements in try block.
2. Git Update type : Merge, rebase, Branch Default
3. How do you run a Jupyter notebook program from a command line or terminal

Python topics :

* Closures and how to use them
* First class functions
* Decorators

Need to understand below:

* getattr()
* \_\_import\_\_(module\_name)
* Class\_
* Module configparser
* \_temp

pytest-capturelog-0.7

PEP-8 style Guide for python coding:

<https://www.python.org/dev/peps/pep-0008/#imports>

<https://pypi.org/project/pytest-html/>

<https://git-scm.com>

Git hub reference for attaching screenshots in pytest html reports:

<https://github.com/rafitur2/Python-Pytest-Selenium-HTML-report-with-multiple-screenshots>

Python Lib folder has all standard modules

Python Site-packages has the installed third party packages.

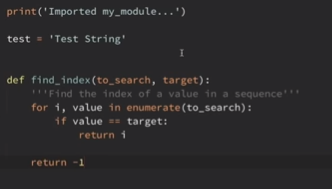
To know the python path use:

Import sys

print(sys.path)

Sample programs to practice:

Import this file as a module and pass a list of subjects(to\_search) and the name to find(target)



1. Write a program for first non repeating character in a string.

s=”aabcc” -> b

s=”abcbad” -> c

s= “abcabc” = -1

Pg-160 covered in Durga

Python Preparation:

Case sensitive: To Python, print, Print, and PRINT are all different things

Spaces matter at the beginning of lines, but not elsewhere in the python code

Variable names can contain letters, numbers, and the underscore.

Variable names cannot contain spaces.

Variable names cannot start with a number.

Case matters—for instance, temp and Temp are different.

The **eval** function is used when we’re getting numerical input from the user. The **eval** function converts the text entered by the user into a number.

The **input** function for getting text from the user.

variable name = **input**(message to user)

**sep :** optional argument called sep, short for separator, that you can use to change that space to something else. sep=''. This says to put no separation between the arguments.

Print (“the sum of 3+4 : “, 3+4, ‘.’, sep=’ ‘)

**end** The print function will automatically advance to the next line, There is an optional argument called end that you can use to keep the print function from advancing to the next line. The end=’ ‘just keeps everything on the same line.

The **is** keyword is used to test if two variables refer to the same object. ... The test returns False if they are not the same object, even if the two objects are 100% equal. Use the == operator to test if two variables are equal.

33 Keywords in Python programming language

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [False](https://www.programiz.com/python-programming/keyword-list#true_false) | [class](https://www.programiz.com/python-programming/keyword-list#class) | [finally](https://www.programiz.com/python-programming/keyword-list#finally) | [is](https://www.programiz.com/python-programming/keyword-list#is) | [return](https://www.programiz.com/python-programming/keyword-list#return) |
| [None](https://www.programiz.com/python-programming/keyword-list#none) | [continue](https://www.programiz.com/python-programming/keyword-list#break_continue) | [for](https://www.programiz.com/python-programming/keyword-list#for) | [lambda](https://www.programiz.com/python-programming/keyword-list#lambda) | [try](https://www.programiz.com/python-programming/keyword-list#except_raise_try) |
| [True](https://www.programiz.com/python-programming/keyword-list#true_false) | [def](https://www.programiz.com/python-programming/keyword-list#def) | [from](https://www.programiz.com/python-programming/keyword-list#from_import) | [nonlocal](https://www.programiz.com/python-programming/keyword-list#nonlocal) | [while](https://www.programiz.com/python-programming/keyword-list#while) |
| [and](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [del](https://www.programiz.com/python-programming/keyword-list#del) | [global](https://www.programiz.com/python-programming/keyword-list#global) | [not](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [with](https://www.programiz.com/python-programming/keyword-list#with) |
| [as](https://www.programiz.com/python-programming/keyword-list#as) | [elif](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [if](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [or](https://www.programiz.com/python-programming/keyword-list#and_or_not) | [yield](https://www.programiz.com/python-programming/keyword-list#yield) |
| [assert](https://www.programiz.com/python-programming/keyword-list#assert) | [else](https://www.programiz.com/python-programming/keyword-list#if_else_elif) | [import](https://www.programiz.com/python-programming/keyword-list#from_import) | [pass](https://www.programiz.com/python-programming/keyword-list#pass) |  |
| [break](https://www.programiz.com/python-programming/keyword-list#break_continue) | [except](https://www.programiz.com/python-programming/keyword-list#except_raise_try) | [in](https://www.programiz.com/python-programming/keyword-list#in) | [raise](https://www.programiz.com/python-programming/keyword-list#except_raise_try) |  |
|  | | | | |

The structure of a **for loop** is as follows:

**for** variable name **in range**( number of times to repeat ):

statements to be repeated

The word **for** must be in lowercase, the first line must end with a colon, and the statements to be repeated must be indented. Indentation is used to tell Python which statements will be repeated.

The statement **range**(1,5) will produce the list 1, 2, 3, 4.

The statement **range**(1,10,2) will step through the list by twos, producing 1, 3, 5, 7, 9.

**range**(5,1,-1) will produce the values 5, 4, 3, 2, in that order.

The only difference is that find() method returns -1 if the substring is not found, whereas index() throws an exception.

Regular Expressions:

re.match(pattern, string, flags=0)

re.sub(pattern, repl, string, max=0)

Regular Expression Patterns

Except for control characters, **(+ ? . \* ^ $ ( ) [ ] { } | \)**, all characters match themselves. You can escape a control character by preceding it with a backslash.

For “.”, we use the escape character “\.”, or else it will match all characters.

Following table lists the regular expression syntax that is available in Python −

|  |  |
| --- | --- |
| **Sr.No.** | **Pattern & Description** |
| 1 | **^**  Matches beginning of line. |
| 2 | **$**  Matches end of line. |
| 3 | **.**  Matches any single character except newline. Using m option allows it to match newline as well. |
| 4 | **[...]**  Matches any single character in brackets. |
| 5 | **[^...]**  Matches any single character not in brackets |
| 6 | **re\***  Matches 0 or more occurrences of preceding expression. |
| 7 | **re+**  Matches 1 or more occurrence of preceding expression. |
| 8 | **re?**  Matches 0 or 1 occurrence of preceding expression. |
| 9 | **re{ n}**  Matches exactly n number of occurrences of preceding expression. |
| 10 | **re{ n,}**  Matches n or more occurrences of preceding expression. |
| 11 | **re{ n, m}**  Matches at least n and at most m occurrences of preceding expression. |
| 12 | **a| b**  Matches either a or b. |
| 13 | **(re)**  Groups regular expressions and remembers matched text. |
| 14 | **(?imx)**  Temporarily toggles on i, m, or x options within a regular expression. If in parentheses, only that area is affected. |
| 15 | **(?-imx)**  Temporarily toggles off i, m, or x options within a regular expression. If in parentheses, only that area is affected. |
| 16 | **(?: re)**  Groups regular expressions without remembering matched text. |
| 17 | **(?imx: re)**  Temporarily toggles on i, m, or x options within parentheses. |
| 18 | **(?-imx: re)**  Temporarily toggles off i, m, or x options within parentheses. |
| 19 | **(?#...)**  Comment. |
| 20 | **(?= re)**  Specifies position using a pattern. Doesn't have a range. |
| 21 | **(?! re)**  Specifies position using pattern negation. Doesn't have a range. |
| 22 | **(?> re)**  Matches independent pattern without backtracking. |
| 23 | **\w**  Matches word characters. (a-z, A-Z, 0-9) |
| 24 | **\W**  Matches nonword characters. (It matches everything except alphanumeric values) |
| 25 | **\s**  Matches whitespace. Equivalent to [\t\n\r\f]. -> matches tab, space, newline |
| 26 | **\S**  Matches non-whitespace. Not matches tab, space, newline (matches alphanumeric, special characters and symbols ) |
| 27 | **\d**  Matches digits. Equivalent to [0-9]. |
| 28 | **\D**  Matches nondigits. |
| 29 | **\A**  Matches beginning of string. |
| 30 | **\Z**  Matches end of string. If a newline exists, it matches just before newline. |
| 31 | **\z**  Matches end of string. |
| 32 | **\G**  Matches point where last match finished. |
| 33 | **\b**  Matches word boundaries when outside brackets. Matches backspace (0x08) when inside brackets. Ex: “do undo” this matches only first “do”, **pattern=(r’\bdo’)** – line starting and after space takes as the word boundary, |
| 34 | **\B**  Matches nonword boundaries. Ex: “do undo” this matches only second “do” in(“undo”) |
| 35 | **\n, \t, etc.**  Matches newlines, carriage returns, tabs, etc. |
| 36 | **\1...\9**  Matches nth grouped subexpression. |
| 37 | **\10**  Matches nth grouped subexpression if it matched already. Otherwise refers to the octal representation of a character code. |

**Try :**

**Import socket**

**re.compile**

Robot Framework preparation :

##### Creating scalar variables

**\*\*\* Variables \*\*\***

${NAME} Robot Framework

${VERSION} 2.0

${ROBOT} ${NAME} ${VERSION}

##### Creating list variables

**\*\*\* Variables \*\*\***

@{NAMES} Matti Teppo

@{NAMES2} @{NAMES} Seppo

@{NOTHING}

@{MANY} one two three four

... five six seven

##### Creating dictionary variables

**\*\*\* Variables \*\*\***

&{USER 1} name=Matti address=xxx phone=123

&{USER 2} name=Teppo address=yyy phone=456

&{MANY} first=1 second=${2} ${3}=third

&{EVEN MORE} &{MANY} first=override empty=

... =empty key\=here=value

**File name is** Search\_flights\_keywords.robot:

**Creating Custom Keywords:**

\*\*\* Settings \*\*\*

Library SeleniumLibrary

\*\*\* Variables \*\*\*

${URL} http://blazedemo.com/

${BROWSER} Chrome

\*\*\* Keywords \*\*\*

Open Home Page

Open browser ${URL} ${BROWSER}

Close Browsers

Close All Browsers

Select Departure City

[Arguments] ${departure\_city}

Select From List By Value xpath://select[@name='fromPort'] ${departure\_city}

Select Destination City

[Arguments] ${destination\_city}

Select From List by Value xpath://select[@name='toPort'] ${destination\_city}

Search For Flights

Click Button css:input[type='submit']

There are available Flights

@{flights}= Get WebElements css:table[class='table']>tbody tr

Should Not Be Empty ${flights}

All keywords we are intending to use should be created in the “Keywords” section.  You can notice that we don’t have any test cases in the file. Such files are considered as resource files

Below is the test case with custom keywords added:

\*\*\* Settings \*\*\*

Resource search\_flights\_keywords.robot

Suite Setup Open Home Page

Suite Teardown Close Browsers

\*\*\* Test Cases \*\*\*

The user can search for flights

[Tags] search\_flights

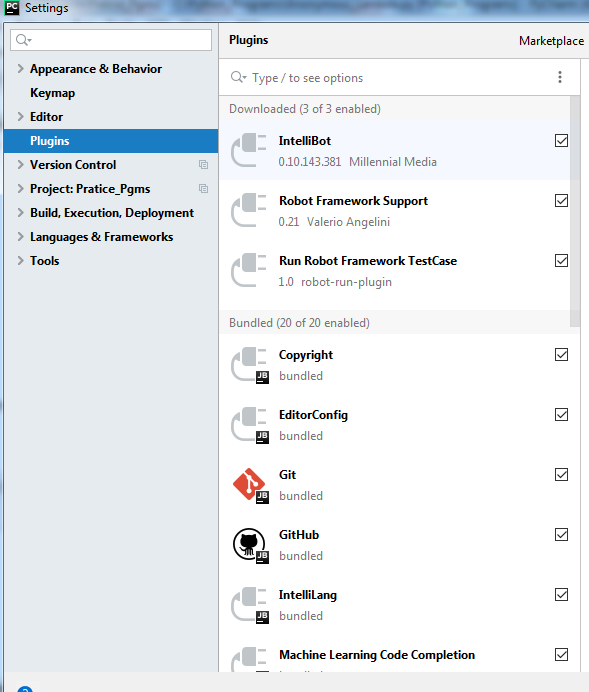
Select Departure City Paris

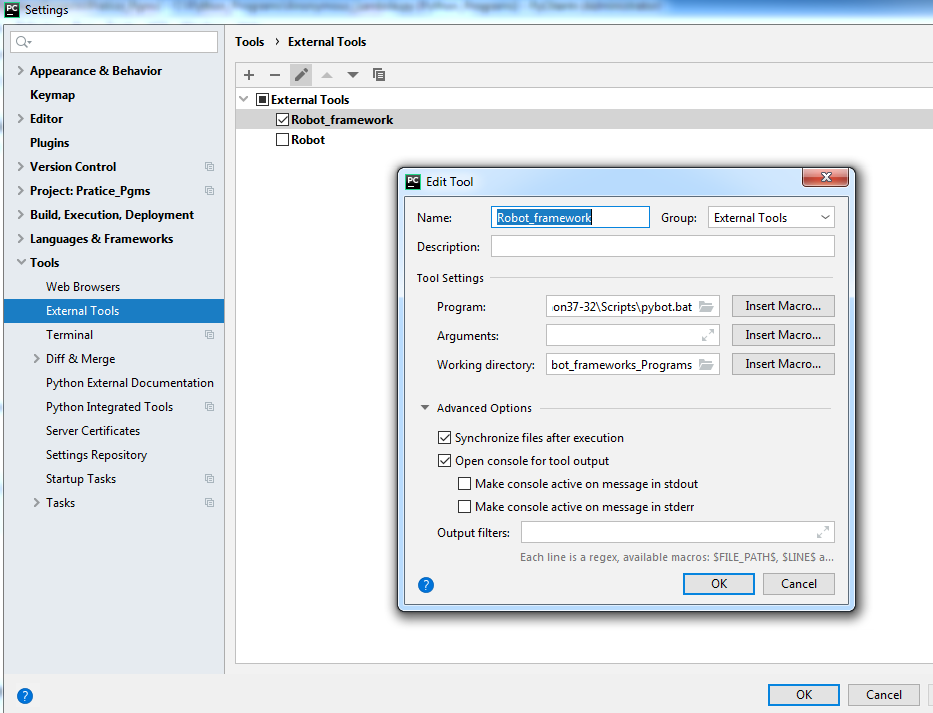
Select Destination City London

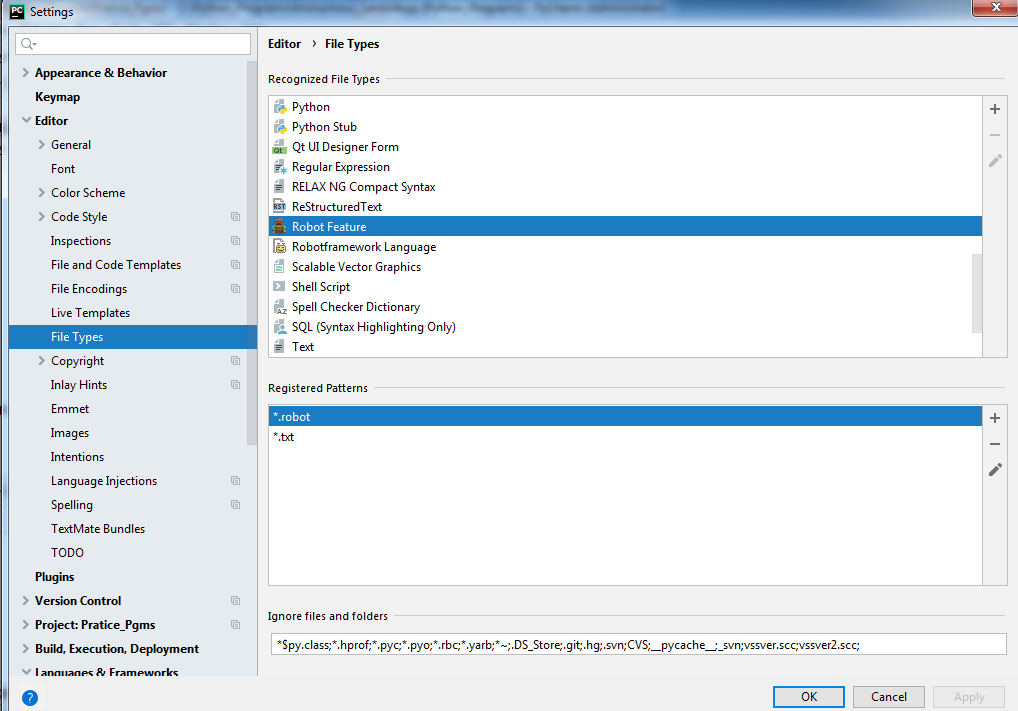
Search For Flights

There are available Flights

**Robot framework installation in pycharm:**







## Arbitrary Arguments

If you do not know how many arguments that will be passed into your function, add a \* before the parameter name in the function definition.

This way the function will receive a tuple of arguments, and can access the items accordingly:

## The pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

**Similarly “break” statement in loops, if you don’t have anything to write inside the loop statement.**

## Keyword Arguments

You can also send arguments with the key = value syntax.

This way the order of the arguments does not matter.

**To access this key=value arguments, need to use \*\* before the parameter in function definition.**

**Difference between Return and Yield :**

**Return, returns the function and destroys all the local variables but Yield saves the point of the last execution.**

**Join method used in Strings:**

The join() method takes all items in an iterable and joins them into one string.

A string must be specified as the separator.

**Syntax:**

string.join(iterable)

The get() method returns the value of the item with the specified key.

dictionary.get(keyname, value)

If the Keyname is not present it will return the value mentioned next to the keyname.

Dic1.get(“Arun”,”Value is not found”)

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| *keyname* | Required. The keyname of the item you want to return the value from |
| *value* | Optional. A value to return if the specified key does not exist. Default value None |

**Dict1[key] -> This will give the value for this key in dictionary Dict1**

**Dict1.get(key) - > This is also same as above but if the key is not found in Dict, you can print some value and the program will not throw error.**

**Dict1.get(key,”Key not found”)**

**Different List Methods:**

|  |  |
| --- | --- |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the first item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

**SETS:**

**The main difference between sets and lists are that sets cannot have a duplicate value but lists have duplicates.**

**Sets cannot be accessed using index position.**

To add one item to a set use the add() method.

To add more than one item to a set use the update() method.

Python has a set of built-in methods that you can use on sets.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [add()](https://www.w3schools.com/python/ref_set_add.asp) | Adds an element to the set |
| [clear()](https://www.w3schools.com/python/ref_set_clear.asp) | Removes all the elements from the set |
| [copy()](https://www.w3schools.com/python/ref_set_copy.asp) | Returns a copy of the set |
| [difference()](https://www.w3schools.com/python/ref_set_difference.asp) | Returns a set containing the difference between two or more sets |
| [difference\_update()](https://www.w3schools.com/python/ref_set_difference_update.asp) | Removes the items in this set that are also included in another, specified set |
| [discard()](https://www.w3schools.com/python/ref_set_discard.asp) | Remove the specified item |
| [intersection()](https://www.w3schools.com/python/ref_set_intersection.asp) | Returns a set, that is the intersection of two other sets |
| [intersection\_update()](https://www.w3schools.com/python/ref_set_intersection_update.asp) | Removes the items in this set that are not present in other, specified set(s) |
| [isdisjoint()](https://www.w3schools.com/python/ref_set_isdisjoint.asp) | Returns whether two sets have a intersection or not |
| [issubset()](https://www.w3schools.com/python/ref_set_issubset.asp) | Returns whether another set contains this set or not |
| [issuperset()](https://www.w3schools.com/python/ref_set_issuperset.asp) | Returns whether this set contains another set or not |
| [pop()](https://www.w3schools.com/python/ref_set_pop.asp) | Removes an element from the set |
| [remove()](https://www.w3schools.com/python/ref_set_remove.asp) | Removes the specified element |
| [symmetric\_difference()](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | Returns a set with the symmetric differences of two sets |
| [symmetric\_difference\_update()](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | inserts the symmetric differences from this set and another |
| [union()](https://www.w3schools.com/python/ref_set_union.asp) | Return a set containing the union of sets |
| [update()](https://www.w3schools.com/python/ref_set_update.asp) | Update the set with the union of this set and others |

**Links:**

**Numpy array:**

<https://www.geeksforgeeks.org/python-numpy/>

# Difference between == and is operator in Python

The **==** operator compares the values of both the operands and checks for value equality. Whereas **is** operator checks whether both the operands refer to the same object or not.

**Python any() Method:**

The any() method returns True if any element of an iterable is True. If not, any() returns False.

* The any() method takes an iterable (list, string, dictionary etc.) in Python.

any() returns:

* True if at least one element of an iterable is true
* False if all elements are false or if an iterable is empty

Python all() Method:

The all() function returns True if all items in an iterable are true, otherwise it returns False.

If the iterable object is empty, the all() function also returns True.

Syntax:

all(iterable)

An iterable object (list, tuple, dictionary)

# floor() and ceil() function in math module Python

floor() returns the integer value of the decimal number supplied to it and returns the integer value rounded off to the nearest value which is going to be lesser than the actual number supplied.

Ceil() is vice versa to floor(),rounded off to the nearest value which is going to be more than the actual number supplied.

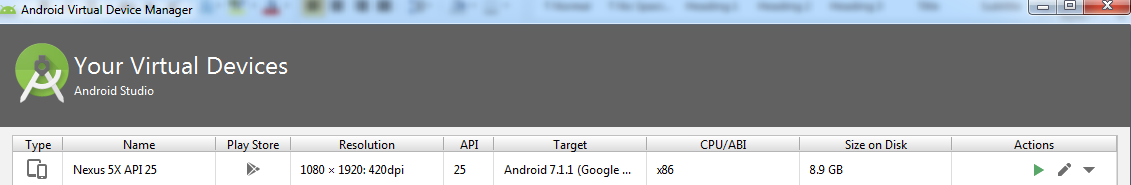
the ‘\_’ variable which is used to hold the value of last expression in interpreter.

There are 5 cases for using the *underscore* in Python.

1. For storing the value of last expression in interpreter.
2. For ignoring the specific values. (so-called “I don’t care”)
3. To give special meanings and functions to name of vartiables or functions.
4. To use as ‘Internationalization(i18n)’ or ‘Localization(l10n)’ functions.
5. To separate the digits of number literal value.

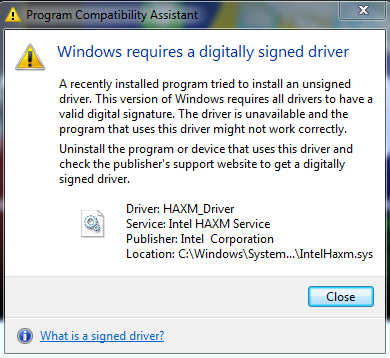
Appium Installation and trouble shooting

* 1. Install Pycharm and Python
  2. Install Andriod Studio and setup type as “Standard” , SDK folder, it will take some time to download all the packages to the SDK folder.
  3. Configure your project in Andriod Studio.
  4. Set the Environment variables to the below path’s :
* SDK\_folder\build-tools
* SDK\_folder\platform-tools
* SDK\_folder\tools
  1. Check in the cmd, “adb” is working, below few commands:
* adb start-server
* adb kill-server
* adb devices
  1. Set-up the Emulator by going to Andriod studio-> AVD Manager, Create Virtual device and select the device you want and download the Andriod Flavor you need.
  2. Start the Emulator by clicking on the Green play icon.



* 1. If the Emulator doesn’t start, below trouble shooting steps:
* Ensure Virtualization is enabled in the system by going to Bios settings.
* Install the Intel HAXM v7.1 available in the link:

<https://github.com/intel/haxm/releases>

* Wipe the data of Emulator by clicking on the drop down in the above screenshot.
* Latest version is having issue with Digital signed certificate on some Windows 7.
  1. The Emulator should be listed in “adb devices”
  2. Download Appium for desktop client : <http://appium.io/downloads.html>
  3. Install Appium.
  4. Vysor : <https://chrome.google.com/webstore/detail/vysor/gidgenkbbabolejbgbpnhbimgjbffefm>

This is used for displaying mobile screens on your computer, available as an extension for Chrome browser.

* 1. You can also try Airdriod.
  2. Download older of version of tools from this link to support “android” command in cmd <https://dl.google.com/android/repository/tools_r25.2.3-windows.zip>

**1. First, get the application package name**

To get the application package name, use:

adb shell 'pm list package -f' |grep calculator

note the APK file and the package name. For example:

package:/system/app/SecCalculator2/SecCalculator2.apk=com.sec.android.app.popupcalculator

the apk file is **/system/app/SecCalculator2/SecCalculator2.apk** and the package name is: **com.sec.android.app.popupcalculator**

**2. Get the launch-able Activity name**

First, pull the application APK's from devices:

adb pull /system/app/SecCalculator2/SecCalculator2.apk

then use **aapt** to get the launchable Activity name:

aapt dump badging SecCalculator2.apk

Look for **launchable-activity** in the results, or use grep to filter it. The result should be similar like this:

launchable-activity: name='com.sec.android.app.popupcalculator.Calculator' label='Calculator' icon=''

On these sample, the launchable activity name is: **.Calculator**

**3. Now, create the DesiredCapabilities**

create your desired capability to launch the installed applications. Make sure to match the **appPackage** and **appActivity** from the steps above. (this sample is using java).

DesiredCapabilities cap = new DesiredCapabilities();

cap.setCapability(CapabilityType.BROWSER\_NAME, "");

cap.setCapability("platformName", "ANDROID");

cap.setCapability("deviceName", "S5");

cap.setCapability("platformVersion", "5.0");

cap.setCapability("appPackage", "com.sec.android.app.popupcalculator");

cap.setCapability("appActivity",".Calculator");

cap.setCapability("noSign", true);

Sikuli Installation:

1. Pip install sikuli
2. This requires JDK and JRE and set the environment variables.

distutils.errors.DistutilsError: Setup script exited with error: Microsoft V

isual C++ 14.0 is required. Get it with "Microsoft Visual C++ Build Tools": http

s://visualstudio.microsoft.com/downloads/

----------------------------------------

Django installation:

1. Pip install Django
2. python -m django –version (3.0.1)
3. C:\Users\Admin>django-admin
4. E:\Web\_App\_Dev>django-admin startproject django\_project1 (Create a new project)

* E:\Web\_App\_Dev\django\_project1>python manage.py runserver

Watching for file changes with StatReloader

Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you

apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them.

December 31, 2019 - 13:28:30

Django version 3.0.1, using settings 'django\_project1.settings'

Starting development server at http://127.0.0.1:8000/

Quit the server with CTRL-BREAK.

1. <http://localhost:8000/admin>
2. <http://localhost:8000>
3. E:\Web\_App\_Dev\django\_project1>python manage.py startapp blog

Run the above command inside the django\_project1.

The above command will create a folder “blog”, inside the blog directory create “urls.py” file

1. Bootstrap link : <https://getbootstrap.com/docs/4.0/getting-started/introduction/>

This allows us to add nice styles and designs to our html pages.

Class, OOPS in Python:

1. The \_\_init\_\_ function will be called first, whenever the instances are created for that class.

Ex:

Class Car:

def \_\_init\_\_(self):

print(“This is from \_\_init\_\_ function”)

ford=car()

Honda=car()

O/P:

This is from \_\_init\_\_ function

This is from \_\_init\_\_ function

1. Python doesn’t have a destructor, it automatically collects the garbage collection.
2. Self is actually the current object, it is similar to “This” in C++ or Java.
3. It is not necessary to use the “self” keyword it can be used anything like “abc”, “xyz” but need to use the same to assign the value to an argument.

By convention “self” is used.

Ex:

Class Car:

def \_\_init\_\_(xyz,speed,color):

print(“This is from \_\_init\_\_ function”)

xyz.speed=speed

xyz.color=color

a=car(200,blue)

1. “Self” is automatically passed when you initialize the class. (mean when you create an instance of a class, “self” is automatically sent as an argument to the class)
2. If multiple \_\_init\_\_ methods are called inside the class, the last called \_\_init\_\_ method is only valid and it overwrites all the “init” methods which is called above.

Basically multiple “init” methods are not allowed inside the class.

Ex:

Class Car:

def \_\_init\_\_(self,speed):

self.speed=speed

self.color=”Blue”

a= Car(200)

The above class example is completely valid, as there is no argument color declared in the init method, but still we are statically assigning value as “Blue”

1. Encapsulation : To protect our data from someone manipulating it.

Ex:

Class Car:

def \_\_init\_\_(self,speed,color):

self.speed=speed

self.color=color

a= Car(200,”blue”)

a.speed= 300

Now the speed argument is overwritten from 200 to 300. We want to prevent from assigning value like this: a.speed=400

For this we use setter and getter for the attributes (function inside the class.)

Ex:

Class Car:

def \_\_init\_\_(self,speed,color):

self.\_\_speed=speed

self.\_\_color=color

def set\_speed(self,value):

self.\_\_speed=value

def get\_speed(self)

return self.\_\_speed

ford=Car(200,”red”)

ford.set\_speed(300)

ford.\_\_speed=400 # even though am trying to override the variable value it will not be set to 400.

print(ford.get\_speed()) -> o/p 300

print(ford.color) # This will give you an error since color is now a private variable.

Print(ford.\_\_color) # This also will give you an error because it is a private variable, can’t be used to print “\_\_color” variable value. (need to use get\_color() method)

We can assign and retrieve the variable value using setter and getter method.

In Python we don’t have private, public methods which is available in C++ or Java.

Instead if we use “\_\_variable\_\_” this makes the data private variable.

So by this, we can’t change the value to the variable outside and we can protect our data.

If we want to change the value need to use “set\_speed” method.

To retrieve the value need to use “get\_speed” method.

This way of restricting the data variable is called Encapsulation.



1. Private member variable: A private member variable is private to the class (means if you try to use it outside the class it will give you error)

You can use the private member variable inside the class or any method inside the class.

Class Car:

def \_\_init\_\_(self,speed,color):

self.\_\_speed=speed

self.\_\_color=color

def public\_method(self):

print(self.\_\_speed)

ford=Car(200,”red”)

ford.public\_method() # this will print the self.\_\_speed which is 200.

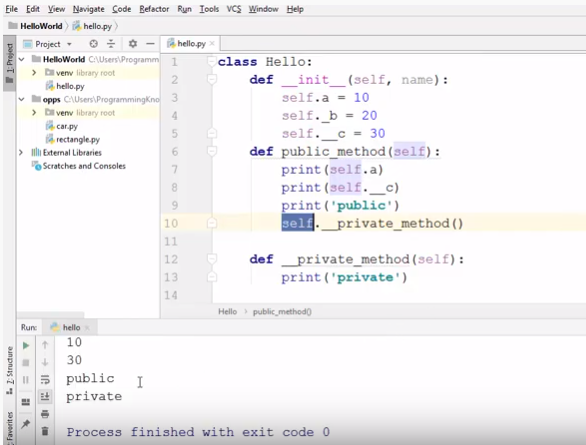
1. Private\_method : To define the private method, use “\_\_” before the method name.

Ex: def \_\_private\_method(self):

Print(“private method)

Same like private member variable, we can’t use private method outside the class.

1. If you want to call any method inside a class, use “self” keyword.



1. Inheritance : The idea is to borrow the class and methods from another class.

Superclass -> Base class or the class from which it is been inherited.

Subclass -> class which is being inherited from base class.

1. You cannot inherit the private members of the super-class into the sub class.
2. Multiple Inheritance : The ability of a class to inherit from multiple classes is called multiple inheritance. In python a class can inherit from multiple class.

To inherit from multiple classes:

Ex : Class name(class1, class2):

Just mention the class names separated from(,) in parenthesis.

1. \_\_init\_\_ function is the first function called when the instance of a class is created. “Super” built-in function.

Ex:

Class Parent:

def \_\_init\_\_(self):

print(“this \_\_init\_\_ from parent class”)

class Child(Parent):

def \_\_init\_\_(self):

print(“this \_\_init\_\_ from child class”)

child=Child()

o/p : this \_\_init\_\_ from child class

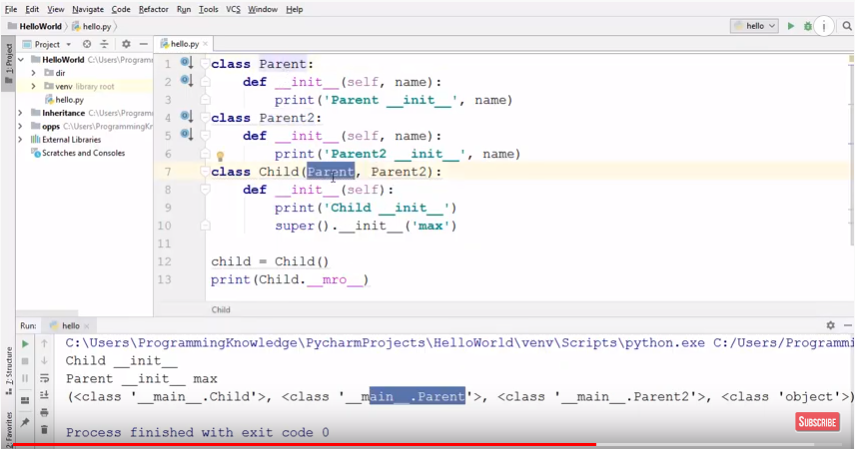
So it is called from the child class, suppose if we want to refer to the parent class \_\_init\_\_ function, we need to use “super().\_\_init\_\_” built-in function, this means we are calling the \_\_init\_\_ method from the super() class, which is Parent class in our example.

\_\_mro\_\_ : Method resolution order.

Ex:

child = Child()

print(Child.\_\_mro\_\_)



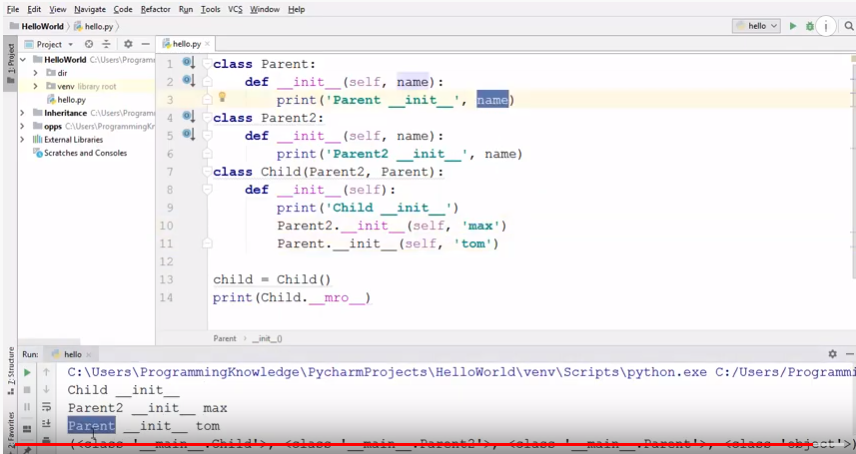
Mro tells the method execution order, it depends on how we inherit the super class, execution order is basically as below:

1. Child method execution
2. Parent method execution, which is first inherited
3. Next second parent method which is executed.

“Super().\_\_init\_\_” refers to the super class.

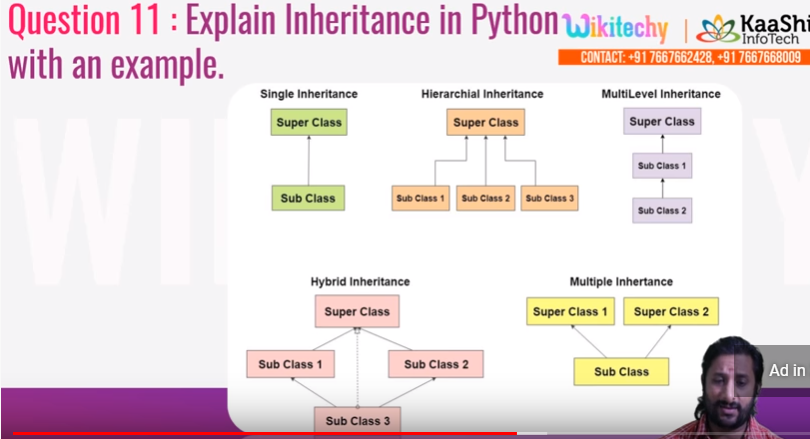
Suppose if we inherit from multiple super classes(base or parent class) then we have to implicitly mention the super class name to use the \_\_init\_\_ methods of those super classes.

Ex:



We are mentioning Parent2.\_\_init\_\_ method and Parent.\_\_init\_\_ method to call their \_\_init\_\_methods respectively.

1. Inheritance types:



Pickling and Unpickling:

The process of writing state of object to the file is called pickling and the process of reading state of an object from the file is called unpickling.

**Pickling - pickle.dump(object,file\_object)**

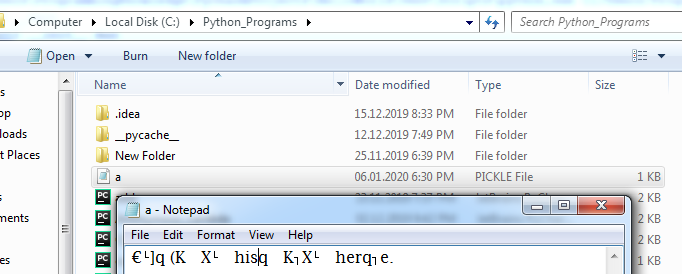
**Unpickling – pickle.load(file\_object)**

We can pickle for list, dictionary

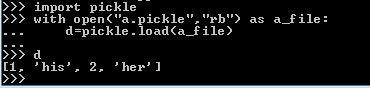
**Pickle in Python** is primarily used in serializing and deserializing a **Python** object structure.

**import** pickle  
a=[1,**"his"**,2,**"her"**]  
**with** open(**"a.pickle"**,**"wb"**) **as** a\_file:  
 pickle.dump(a,a\_file)

This creates a pickle file as shown below:



Unpickling:



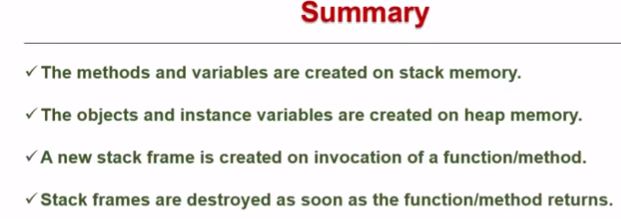
We are reading the “a.pickle” file.

Memory Management in python:

Two types of Memory:

* + 1. Heap memory
    2. Stack memory

Objects are created in Heap memory whereas references are created in Stack Memory.



Garbage collection is a method to clean up the dead object.

Monkey patching: the term monkey patch refers to dynamic (or run-time) modifications of a class or module.  
In Python, we can actually change the behavior of code at run-time.

Method Overloading:

In python, to make the same function work differently, i.e, as per the arguments

Web scraping:

Web scraping is method of extracting the large amounts of information which is available on the websites and saving it onto the local machine or onto the database.

In python, following or some modules for web scrapping:

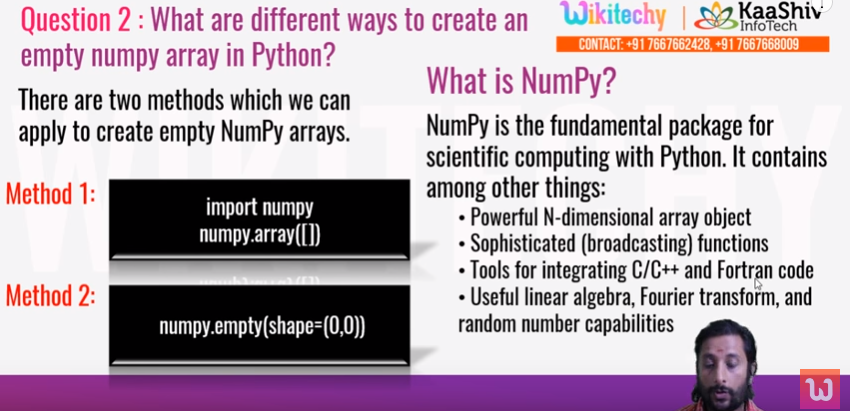
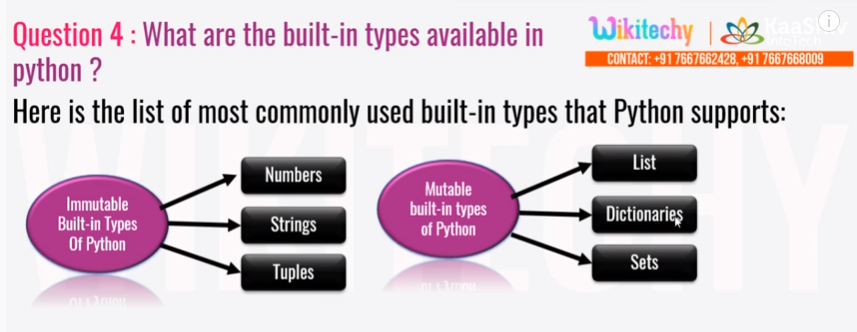
1. Urllib2
2. Scrappy
3. Pyquery
4. BeautifulSoap etc.

Some Interview ques:

1. To debug a python program command:

python -m pdb decorators\_function.py

Press ‘n’ to go to next line for debugging.

1. 
2. 

* double underscores is shortened as dunder

Pytest

run with following command:

pytest test\_math\_func.py -v or py.test

suppose the file name is math\_func.py, still you can run pytest using command : pytest math\_func.py, if the file name is starting with test word, then you can simply use py.test

option -v : for verbose logging enabled.

#Pip install beautifulsoup4

* scope of pytest : Session, Module

@pytest.fixture(scope=’session’)

@pytest.fixture(scope=’function’) # Default scope

@pytest.fixture(scope=’module’)

@pytest.fixture(scope=’class’)

* Fixture can use fixtures too.
* pytest.skip(“wanted to execute later”) -> you can skip some test for later execution.
* Pytest marks – Run tests based on markers
* @pytest.mark.mymarker
* Python enhancement proposals – 8 (PEP-8) – https://pep8.org