**Python Programming**

(Basics)

**(notes**)

**What is python?**

Python is object oriented language and is independent language created by Guido Rossum 1989. Many large companies use python such as NASA, Google, YouTube, Facebook etc.

**Core Concepts:**

**1): String:** series of character.

**2): Expression:** piece of code provide you a value

**3): Variable:** like used in mathametics for example (price=5)

4): Integer: number like 1 3 3.

5): floating number:number with decimals.

6): boolean values: true or false

7): index: these are the sequence of the charactes, we use [] to write it but if we use 0:3 so will write letter beetween 0:3 (3 excluded) like in the below example it will write My

For example:

Variable= My name is Maaz.

Print(Variable[1])

Result:

It will print the first letter of the sentence but if we write -1 so it will be written the last character.

**Data types (classifications/ categorizing the data):**

**1): integer**: as the name suggests number

**2): String:** text

**3): Float:** decimals

**4): Boolean:** yes or no

**OPERATORS IN PYTHON:**

**1**): In: this operator is used to check that either the given character is present in the variable or the given string. This will only provide a boolean value.

For example:  
“saim” in name

2): + - \* / // % \*\*: these are arthemitic operation same as use in maths.

3): += -=: these are augmented operator

**4):** and: logical operator in this both the conditions have to be true

5) or: logical operator in this one of the conditions have to be true.

6) not: logical operation condition have to be false.

7) < > >= <= == !=: these are comparision operator used in if statement to provide a condition

**Conditional STATEMENTS IN PYTHON (helps to control the flow of a program. condition must be evaluate either True or False):**

**Syntax:-**

if condition:

statement

elif:

statement

else:

statement

**Common operators :-**

< ---- Less than --> if a < b

<= --- Less than equal to --> if a <= b

> --- greater than --> if a > b

>= --- greater than equal to --> if a >= b

== Equal Equal --> if a == b

!= Not Equal --> if a != b

! Not --->!a

**Simple if condition example:-**a=input("enter a number a")

b=input("enter a number a")

c=input("enter a number a")

if a > b and a > c:

print(f"{a} is big number")

elif b > c and b > a:

print(f"{b} is the big number")

else:

print(f"{c} is the big number")

**Nested if condition example:-**

if a>b:

if a>c:

print("a is greater number")

else:

print("c is big number")

else:

if b > c:

print("b is big number")

else:

print("c is the big number")

**Some string operations:**

1): len: calculate the len of a string

2): concatenate:

3): upper(): will turn variable into uppercase

example:

print(variable\_name.upper)

4): capitalize(): first letter become capital case  
5): casefold(): first letter become small case

6): string.startswith():

7): delete or modify

8): center

9): strip(‘\*’) **(very useful)**: will remove the \* sign from the string, will not strip middle value

10): rstrip(‘Saim‘)

11): lstrip(‘Saim‘)

12): split(): will split each character

13): join: used to replace character

example:-

mykoint=' '.join(mysplit)

>>> print(mykoint)

My name is Saim Talha

OR

print(" ".join(mysplit))

14): .replace(‘old’,”new”)

print(name.replace("\n",""))==> example

15): sorted(var\_name): this will sort each and every task

**Note (could be interview question):** if want to get to know that which possible string operations could be used so use this code dir(variable\_name) **and** you can delete the whole string by del operation but can not modify the string

script: (this script will make readable the string)

input=”abdealiabdealiabdeaeliabdeali”

pattern=”abdeali”

re=input.count()

>>>print[pattern]\*re

**How to reverse a string?**

print(name(::-1)) or print("".join(reversed(name)))

**Data structures:-**

Data structures are the set of data elements that produces a well organized way of storing and organizing data

**There are four data structures in python**

**1. set --→ {} → (not very important)**

set={1,3,5,5}

**example:-**

print(set(abd)) → to convert list to set

**2. List (**it is mutable means data could be change, it is ordered sequence of elements seperated by [] sign**) → []**

**example:-**

abd=[1,3,5,8] ==> basic example

indexing

1 3 5 8

0 1 2 3 ==> positive indexing

-4 -3 -2 -1 ==> negative indexing

print(abd[1])

>>print(abd:1) ==> to call multiple values

**>>** print(abd::2) ==> to have multiple values with gap

>> print(abd::-1) ==> will reverse the values

>> print(abd.index(8)) ==> to check an index value of a number

>> del abd[0] ==> will delete the first value in the list this shows list is mutable

>> del abd[:10] will delete first 11 values

>> del abd[:] will delete all data

>> var.remove(data\_needs\_remove): index number will not work

>> abd.pop() : will remove the last values

>>> list(range(1,11))

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>> print(min(abd))

>> print(max(abd))

>> print(sum(abd))

>> print(id(abd)) → to get the memory id of the variable

>> print(list1+list2) → to **concatenate**

**>>** abd.**append**(var\_name) → to **append** and add a value into a **list** at the end

[4, 5, 7, [5, 10]] → will get this output when append

>> abd.exten(var\_name) → will get this output slightly different with append  
[4, 5, 6, **5, 10**]

>> print(sorted(abd)) → so get the list in ascending order

>> abd.sort(reverse=True) → to get in descending order

**Note:**

there is a very useful function in python also called as **zip function** it will combine two lists one after another as shown in the example

file\_data=list(zip(courses,prices))

**o/p:-**

**[**Python,"3,500 PKR"  
Sysops,"22,000 PKR"  
scada,"7,500 PKR"**]**

**nested list example:-**

abd=[1,2,4,[5,6],9,10]

>> print(abd[3][0]) ==> will print 5

**3. Dictionary → {} (very important)**

**Defination:-**  
can be implement in various ways , also known as associate arrays.

Dictionary is pair with unique keys

Dictionary are optimized to retrieve values when the key is known. Dictionary are used to stored values in **key value pair**, key:value

it is collection of orders which is not changeable and not duplicate.

**Different data types**

person = [12,3,5,7]#LIST  
person =(12,4,5,6)#Tuple  
person = {12,4,5,6,7}#Set  
**person= {"name":"abd","age":67}**name → key

abd → value

age → key

67 → value

**Basic used and example:-**

dict1={"uname":"abd","pass":"abd123"}

**How to access the value by keys**

**>>>** print(dict1[uname]) → abd

>>> print(dict1.get('Pass')) → how to handle your exceptional handling while keys is not found in to your dictionary (**interview question**)

**How to assign value in a dd**

person={'name': 'saim', 'city': 'karachi'}

person[‘country’]=’pakistan’

**after**

{'name': 'saim', 'city': 'karachi', 'country': 'Pakistan'}

can also change the existing key value

**Copy**

if copy the exact dictionary so memory allocation would be same

**How to get keys of a dd**

>>> print(person.keys())

dict\_keys(['name', 'city', 'country'])

**How to get all the values**

>>> print(person.values())

dict\_values(['saim', 'karachi', 'Pakistan'])

**How to get all the items**

>>> print(person.items())

dict\_items([('name', 'saim'), ('city', 'karachi'), ('country', 'Pakistan')])

>>> print(type(items))

<class 'dict\_items'>

**How to clear a Dds**

>>> person.clear()

>>> print(person)

{}

**How to update via variable**

>>> person = {"name":"ABD","age":44,"city ":"Bangalore " ,"country":"India"}  
>>> person.update(person1)

**POP**

>>> cred.pop('pass') → remove the key value pair

**DEL (delete)**

>> del cred → delete all the dictionary data

>>> del cred['server1'] → will delete a particular key value pair

**Convert list to dictionary**

abd1=['one','two','three']

abd2=[1,2,3]

>>> mydata=**zip**(abd1,abd2) → zip is the keywor that convert it into hexadecimal

>>> print(dict(mydata))

{'one': 1, 'two': 2, 'three': 3}

**From keys**

>>> mydata=dict.fromkeys(abd1,values)

{'a': 'vowel', 'e': 'vowel', 'o': 'vowel', 'i': 'vowel', 'u': 'vowel'}

**Set default value**

>>> username={}

>>> username.setdefault("username","abd")

'abd'

>>> print(username)

{'username': 'abd'}

**To get all types keys and values**

{1: 3, 5: 10}

>>> print(dict1.keys())

dict\_keys([1, 5])

>>> print(max(dict1.keys()))

5

>>> print(max(dict1.values()))

10

>>> print(min(dict1.values()))

3

**To get sorted output**

>>> print(sorted(cred))

['pass', 'server1', 'uname']

**Nested Dictionary**

myinfo = {

1 : {'uname':'saim','pass':'abd123@','server':'Linux'},

2 : {'uname':'maaz','pass':'abd123@','server':'Windows'},

3 : {'uname':'sohaib','pass':'abd123@','server':'RHEL'}

}

**>>>** print(myinfo[3])

>>> print(myinfo[3]['pass'])

**interview question:-**

**what is difference between python2 and python3 for dictionary data when you print**

it will show unordered data when might using python 2

**4. Tuple --→ ()**

Tuple Data Structure define as ()

Tuples are used to store items in single variables

tuples are immutable **(whereas list is mutable → hot interview question)**, this means the data cannot be changed, tuples are more memory efficent then lists,

**Basic examples:-**

abd=(1,5,6)

abd1=(1, 3, 5, [1, 'SAIM', 4], 6, 7)

abd=(1) → not a tuple abd=(1,) → a tuple

**>>>** abd[1]="abd" → could not support which mean tuple is immutable

>>> abd1[3][2]=”SAIM” → will work because list is used

>> print(min(abd))

**>>** final=tuple1+tuple2

>> **print((abd,)\*2) →** to get the tuple multiply by two

**slicing**

>>> print(abd[::-1]) --→ for descending order

**How to convert to tuple:-**

>> tuple(list\_name)

**count a value number of times repeated:-**

abd.count(4)

**Note:-**

**empty** tuplewillhave always **fasle boolean value**

**could not** delete specific value in a tuple can just only delete the **whole tuple**

**Functions:**

a function is a block of code which only runs when its called.

It is also called method or procedure

Functions also helps us to break out program into smaller and modular chunks and its avoid repeatation and makes code being reusable.

**Functions syntax:-**

def functionsname():

statement

function-name() → this is to call the function

**simple example:-**

def abd(): print("Saim") print("talha bhai")print('==================this is the first call====================')abd()print("=================this is the second call====================")abd()print('==================this is the third call===================')abd()

**Example 2:- →** this example will also pass parameters abd

def myban(abd): print('\*'\*30) print('\t\t' + abd) print('\*' \*30)myban('This is example of functions')

**Outputs:-**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

This is example of functions

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Example 3:- →** ali value is outside accessed

def first():  
 print('My first functions')  
 print('My first function value is ' , ali)  
  
ali=2  
first()

O/p:-

My first functions

My first function value is 2

**Note:-**

don’t use the functions name as same as predefined module

**Types of functions:-**

1): Built in functions

2): defined functions

**Using Parameters examples:-**

def add(a,b): c=a+b

print(rc)

def mycode(): a=int(input("Enter the number")) b=int(input("Enter the number")) add(a,b)

**Variable length argument example in functions**

Simple example

def mycode(server,ip,OS):  
 print(f"Server is : {server}\n IP is {ip} \n Os is {OS}")  
  
mycode("IBM","192.168.100.1","Rhel")

or this

def mycode(\*data): # ===> main difference is here  
 print(f"Server is : {data[0]}\nIP is {data[1]} \nOs is {data[2]}")  
  
mycode("IBM","192.168.100.1","Rhel")

**“ \* “ argument example:**

def mycode(\*mydata):  
 for i in mydata:  
 print(type(i))  
 return None  
  
mycode(4,5,6)

**“ \*\* “ argument example:**

def mycode(\*\*mydata):  
 for k,v in mydata.items():  
 print(k,v)  
  
mydict={'server':'IBM','IP':'192.168.100.12','OS':'Rhel'}  
mycode(\*\*mydict)

what is difference between the above two examples?

Key Differences:

* Argument Type: \* collects positional arguments, while \*\* collects keyword arguments.
* Data Structure: \* creates a tuple, while \*\* creates a dictionary.
* Access: \* uses indexing to access elements, while \*\* uses keys to access values.
* Usage: \* is often used for functions that can accept a variable number of positional arguments, while \*\* is often used for functions that can accept a variable number of keyword arguments or for passing dictionaries to functions.

**Lamda Function (example):**

result=lambda ali,kazim:ali+kazim  
print(result(10,20))

**\_ \_ name\_ \_** functions use and example:

this is use so the unwanted data couldn’t be seen

example

import os  
  
def mycode():  
 print("This is my code function ")  
  
  
  
if(\_\_name\_\_=='\_\_main\_\_'):  
 print(os.listdir(os.getcwd())) --|this was the global function   
 print("My global print") --|i wasn’t like to call

**Modules:-**

Some times you might write some code useful across several programs. It is possible to abstract out that info separate file and import the files into program that you want to use code.

Python also make it easy to use other people modules

**3 types of Modules:-**

**1)** Own module

**Basic example**

###########lname.py###########

lname=”saim”

**########script.py############**

**from mylname import \* →** very useful

print(lname)

**Warning:-**

Don’t name your module as the same as your default built-in modules

**2)** Built in modules

**basic example**

import os

os.system("dir")

>>> dir(os)

**3)** Third-party modules

**Basic example:-**

>>> pip install pandas →it will install latest version

>>> pip install pandas == 1.3.4 → for specific version

>>> pip install --upgrade pandas → to upgrade the pandas version

>>> import pandas

>>> selenium.\_\_version\_\_ → to check version'4.31.1'

**Notes:-**

>>> pip install pandas →it will install latest version

>>> pip install pandas == 1.3.4 → for specific version

>>> pip install --upgrade pandas → to upgrade the pandas version

**Loops in Python (very importan):**

Loops helps you execute a block of code repeatedly.

**For Loop:**

**Syntax**

for <variable> in <range or list ,dictionary or whatever> :

statement

**Simple examples**

for i in range(1,10): print(i)

mylist=['Saim','Talha','Banglore']for i in mylist: print(f"my list value is : {i}")

Example 2 (to print values in a dictionary):

myinfo={"name":"saim","city":"karachi","age":12}  
  
for k,v in myinfo.items():  
 print(f"my key is {k}")  
 print(f"my value is {v}")

range:

print(list(range(0,11,3)))

**Situation:-**

Syntax: newfiles = [expression for variable in iterable if condition == True]

example:

newfiles = [i for i in myfiles if i != "abdeali.csv" ]

**While Loop**

**while <true condition>:**

**statement**

**basic example:-**

i=10  
  
while i >= 0:  
 i=i-1  
 print(i)

**Control statements**

**break** command is used for terminating a loop like in this example

i=1while True: print(i) i=i+1 if i >= 5: break

**Pass Statement:**

it is a null statement, when the python interpreter comes across pass statements, It does nothing and ignores

Pass statement is use as a placeholder inside loops, functions, class, if statement implement later.

**IMPORTANT FUNCTIONS:**

1): print(“”):this will write something

**advance print example**

#print(f"your string value is : {strings} \nyour number is {str(integers)} \nyour float is: , {str(floats)}" )

#print(“your string is {} \n your number is {} ”.format(1variable\_name, 2nd variable-name))

#print("your string is {}\n your number is %s\n your digit is %d \n your float is %f )"%(strings,integers,floats))

2): input(): this is used to get an input from the user

3): int(): this will convert the string(or any type conversion) into an integer

4):float(): convert any type into float type means number with decimal

5):bool(): this will convert any type into boolean.

6):type():this will display the type of the variable.

7: f’’:to write formatted strings {} will assign the values dynamically.

8):len(): this will tell you the length of a particular variable of a string.

9): use the .(dot function) to convert a string into various types like (upper case and a lower case) these are some examples.

name.upper()  
name.lower()  
name.find()  
name.replace()

10): round():this will round of the value like in maths.

11): abs(): it will give you the absolute value which is positive

12):break: it is used to break a function.

13): range():this will create a range of numbers like range(10) will give you numbers from 0 to 10. For xample

If type: range(2 ,20):will get the number started from 2 till 20

If type: range (2,20 ,2)will get the number by skipping one and another

14): type(): tell about the class of the data

15): eval(): will define the datatype by his own

**IMPORTANT NOTES:**

1. Python excecute data from top to bottom like if I had written price=10 in first line and price=5 in second line and ask them to print it so it will print 5 not 10.
2. Python is a case sensitive language.
3. We use \_ (lower case) to separate the words
4. In python +-\* are used to add subtract and multiply.
5. By default if write some code in python so it will be treated as a string.
6. If you are writing a string for a variable so it will be ended to the second inverted comma to solve it you can write double quotes instead of single and single instead of double.
7. Can not concatenate float and integers
8. we use \n if want text in the next line
9. We use ‘’’ ‘’’ to write a string in multiple lines.
10. We can use [:] to clone our string.
11. Python also followo the basic BODMAS rule

10): In python we modules that contains reusable functions for example math module for advanced mathametics.

11): input function always provide you with a string.

12): while loops and if statements optionally have the else statement.

13): lists are another type of data used in python and they are also objects and can be changed if needed.

14): tuples are also another data type but are immutable mean that once its created so it cannot not be changed better for memory point of view.

15): we use () to create a tuple and [] this to create a list

16): python is a dynamically typed programming language unlike other languages like Go and Java.

17): python has a very easy syntax.

18): Always declare a variable in lower case

19): Modules are called as a group of function.

20):Pakages are collection of modules.

21):DevOps engineer usually consume packages.

22): “pip” is the package installer for python

23): “**PYPI**”(python package installer) is a platform where you can get many packages

24): virtual enviroment create a logical seperation of the packages install in different projects so any conflict doesn’t occur.

25): should use 0:2 to get the two first value of a variable when using print function

**KEY WORDS IN PYTHON:**

1):def: this is used to defined a function.

Syntax: def FunctionName ():

2):import: to import a module.

3):return: this keyword return a value of a variable.

4):

**Exception handling (** fighting with error**):  
types of error**

1, Syntax error → couldn’t be handle

2, Logic error → couldn’t be handle

b=100   
a=100  
  
if a>b:  
 print("a is big")  
  
else:  
 print("b is big")

**3): Exception handling**

* happen when the program is been executing,
* python generates exception that could be handled
* You can define your own error
* Also called as runtime error

**Exception handling syntax:-**

try:

{you can paste your program here}

except:

{you can print your own error here}

**Better handling**

try:

{you can paste your program here}

except Exception as abd:

print(“something having an issue”,abd)

Note:-

* could also use **else** statement if the program is working successfully as shown:-

try:  
 import os  
 X = "ABC"  
 Y= "BCD"  
  
except NameError:  
 print("Something having issue in your variable, plz check your variable define or not")  
  
else:  
 print("this is an else statement")

* there is also a **finally statement** unlike else, it will run no matter program is executed successfully or not

try:  
 import os  
 X = "ABC"  
 Y= "BCD"  
  
except NameError:  
 print("Something having issue in your variable, plz check your variable define or not")  
  
else:  
 print("this is an else statement")

finally:  
 print(“This is program ending”)

**Some common types of errors**

**1): Name error:-**

**example program**

X="ABD"  
print("MY value is" + X)  
print("MY value is" + Y)

O/p:-

NameError: name 'Y' is not defined

**how to handle this error:-**

try:  
 X = "ABD"  
 print("MY value is" + X)  
 print("MY value is" + Y)  
except Exception as abd:  
 print("Something having issue",abd)

O/p:- → this is a standard output

MY value isABD

Something having issue name 'Y' is not defined

**2): Index Error**

**example of an index error program:**

mylist=[1,2,3,4]print(mylist[4])

**o/p:**

IndexError: list index out of range

**how to handle this error:**

example program

try:  
 mylist = [1, 2, 3, 4]  
 print(mylist[4])  
except Exception as e:  
 print("Something issue",e)

O/p:-

“Something issue list index out of range”

**3): Import error**

**example of module error program:**

import oss

**o/p:**

ModuleNotFoundError: No module named 'oss'

**how to handle this error:**

example program:

try:  
 import oss  
 os.system('dir')  
except Exception as e:  
 print("somehting issue",e)

**4): Zero division error**

error example program

a=10  
b=0  
result= a/b  
print(result)

**O/P:-**

ZeroDivisionError: division by zero

**How to handle this error**

try:  
 a = int(input("Enter a number: "))  
 b = int(input("Enter a number: "))  
  
 result = a / b  
 print(result)  
except Exception as e:  
 print("Something issue",e)

**Handling more than one error at one time:**

**Some frequently used functions in DevOps:**

1. String\_name.split(“/”) : this will split a string into substring. Like it will change the “hello world” string into “hello” world string.
2. “+” : this is called concatination (when add two or more strings.)
3. txt.strip():used to remove extra whitespaces and specified characters from the start and from the end of the strip irrespective of how the parameter is passed
4. regular expression module is a very good module in python to specifies a set of strings that matches it and it is very easy.

**Escape Sequences(sequence of characters that has special meanings):-**

**1): \n:** for a new line

**2): \:** so special characters could be ignored

for example:-

C:\\users\\newfile

**3): \t:** for horizontal tab

**Regular Expressions (regex):-**

A regex is a sequence of characters that forms a search programs.

A regex is special sequence of character that help you match or find other string. In UNIX environment regex is widely used nowadays.

**SomeRegex Operations :-**

**1.** **re.match (**by default it will consider the values that starts from the 0 index weather is multi-line or single**)**

**2. re.search** (only consider the first search)

Example

import remytext="My 192.168.1.8 jboss server ip address is \nMy Docker server ip address is 192.168.1.11 \n My broadcast addres is 255.255.255.255 \n My invalid address 192.168.1234.12345"pattern=r"\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"myfind=(re.search(pattern,mytext))print('We found the string from the pattern',myfind)print('We found the string from the pattern',myfind.group())print('starting index',myfind.start())print('Ending index',myfind.end())print('Search Length is: ', myfind.end() - myfind.start())

**o/p:-**

We found the string from the pattern <re.Match object; span=(3, 14), match='192.168.1.8'>

We found the string from the pattern 192.168.1.8

starting index 3

Ending index 14

Search Length is: 11

**Example 2:-**

import remytext="My 192.168.1.8 jboss server ip address is \nMy Docker server ip address is 192.168.1.11 \n My broadcast addres is 255.255.255.255 \n My invalid address 192.168.1234.12345"pattern=r"ABD\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"myfind=(re.search(pattern,mytext))if myfind: print('We have find some string from pattern',myfind.group())else: print("sorry no value found")

**o/p:-**

**3. re.findall (**search from all over the string**)**

**Syntax:**

re.findall(pattern, txt)

**some flags of regex:-**

re.I → will ignore case

re.M → to not the multiline

Example (using both)

import re  
myword="""  
My jboss server ip address is 192.168.1.8  
My Docker server ip address is 192.168.1.11  
My broadcast addres is 255.255.255.255  
My invalid address 192.168.1234.12345  
my invalid address is 0..1.3  
"""  
pattern=r"^My"  
print(re.findall(pattern,myword,re.M | re.I))

**o/p:-** ['My', 'My', 'My', 'My', 'my']

**Note:-**

total could give three parameters if greater then three so use | sign

**Basic Example**

import remy\_word= "My name is saim from karachi, We are learning python and this course we will learn two versions of python, We will learn python selinuim in this course also"pattern="We"print(re.findall(pattern,my\_word))

**if want to get a specific integer so we use brackets**

Example (in this example total 3 things is being demanded in pattern)

import re  
  
my\_word= "My name is saim from karachi, We are learning python and this course we will learn two versions of python such as python2 and python3, We will learn python selinuim in this course also"  
pattern='python[' '23]'  
print(re.findall(pattern,my\_word))

or (this pipe sign act as an **or** word)

import re  
  
my\_word= "My name is saim from karachi, We are learning python and this course we will learn two versions of python such as python2 and python3, We will learn python selinuim in this course also"  
pattern='python2|python3|python|Python'  
print(re.findall(pattern,my\_word))

**Patterns Key words**

‘\S’ or ‘\w’ (it will include a-z A-Z 0-9**):**

in this example only 5 letters would be printed

import re  
  
my\_word= "My name is saim from karachi, We are learning python and this course we will learn two versions of python such as python2 and python3, We will learn python selinuim in this course also"  
pattern='\w\w\w\w'  
print(re.findall(pattern,my\_word))

**O/p:-**

['karac', 'learn', 'pytho', 'cours', 'learn', 'versi', 'pytho', 'pytho', 'pytho', 'learn', 'pytho', 'selin', 'cours']

\**d →** will only consider digits

. → will consider single character including spaces

\. → find only dot in your string

Example

import re  
  
my\_word= "My name is saim from karachi, saimtalhaferoz@gmail.com We are learning python and this course we will learn two versions of python such as python2 and python3, We will learn python selinuim in this course also"  
pattern='\@\w\w\w\w\w\.\w\w\w'  
print(re.findall(pattern,my\_word))

O\p:-

[‘@gmail.com’]

? → starting the pattern

Example

import remy\_word= "My name is saim from karachi, saimtalhaferoz@gmail.com We are learning python and this course we will learn two versions of python such as python2 and python3, We will learn python selinuim in this course also"pattern='python?'print(re.findall(pattern,my\_word))

**o\p:-**

['python', 'python', 'python', 'python', 'python']

**\b** ==> at the end and the begin of the string

pattern='\\bcourse'

**Example 2 (**this will show 3 repeatations of last character **)**

import remy\_word= 'This is course coursessss coursesss'pattern='courses{3}\\b'print(re.findall(pattern,my\_word))

**o/p;-**

['coursesss', 'coursesss']

**\B ==>** ignore spaces unlike \b could say both are opposite of each other

**\t →** consider **tab** only

**Example**

import remy\_word= 'Saim saim saim saim Saim saim saim saim'pattern='\\t'print(re.findall(pattern,my\_word))

**o\p:-**

['\t', '\t', '\t', '\t', '\t', '\t']

**Real time Example** (extracting ip addresses):

pattern='\d{1,3}.\d{1,3}.\d{1,3}.\d{1,3}'

**or**

pattern='\d+.\d+.\d+.\d+

**note:-**

+ means zero or more time

? means zero times or none like 0..1.1

**4. re.sub** (use to replace pattern and should have three arguments)

**Example:-**

import remyword="My name is saim talha from karachi, We are learning python course with two version such as python 2 and python3"pattern='\\bcourse\\b'print(re.sub(pattern,'COURSE',myword))

**o/p:-**

My name is saim talha from karachi, We are learning python **COURSE** with two version such as python 2 and python3

**6. re.split** (will remove all the match pattern means it will remove all the data that is like the patter as shown In the example):

Example:-

import reabd='192.168.1.8 jboss server ip address is \nMy Docker server ip address is 192.168.1.11 \n My broadcast addres is 255.255.255.255 \n My invalid address 192.168.1234.12345'pattern="\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"my\_split=(re.split(pattern, abd))print(my\_split)for i in my\_split: print(i)

**o/p:-**

jboss server ip address is

My Docker server ip address is

My broadcast addres is

My invalid address 192.168.1234.12345

**7. re.compile** (used to compile regular expressions or string to object)

Example

import re  
myword="We are learning automation python course, in devops course python is very important"  
pattern=re.compile('course')  
print(type(pattern))  
  
print(re.findall(pattern, myword))

**o/p:-**

<class 're.Pattern'>

['course', 'course']

**Note:-**

comparatively more faster then other methods that why **recommended to use** in **production base** environments z