**Introduction To Python**

**Why python?**

* Written by Guido van Rossum in 1990.
* Was named after BBC Show **Monty Python's Flying Circus**
* Practical uses of python.

**Advantages of Python**

* **The Code looks similar to how Humans think.\*\***
* **Python is free and open-source** - open source means that's it's source code is available to public . we can download it , change it ,use it and distribute it. This feature is called FLOSS (Free and open source software.)
* **Portable** - we does not need to change the code that's written in windows operating system to run it on mac or any other operating system.
* **It is well supoorted and has numerous online content.**
* **It is Dynamically typed** - meand the type of value is decided at runtime , this is why de don't need to specify the data type while decalaring it.
* **It has many powerfull libraries for data manipulation and analysis , data visulatization.**
* **Python has Machine Learning.**

**Downloading and Installing**

* It can be downloaded from Python official website download section. - [**https://www.python.org/downloads**](https://www.python.org/downloads)
* A executable file (.exe) will be download and upon running it python will be installed in machine.

**File Extension -**

**.py**

**IDE (Integrated Development Environment) - Is a the Text Editior for writing the Program ( Code ).**

**Python Distributions**

* Python software foundation does regular releases of python on its offical webstie . we call them standard distribution.
* Other python distribution are downloaded as per person's convenience , they also have many 3^rd party libraries pre installed.

Examples of python distribution -

* Anaconda from continuum analytics
* Pocket python
* Portable Python - run python from USB device.
* Active python from active state.

**Anaconda Python Distribution**

* We can download it from - <https://www.anaconda.com/distribution/> , use graphical installer
* A .exe file will be downloaded ,run it and Anaconda will be installed.
* Run anaconda distribution.

**Lets have a brief introduction to Anaconda navigator and Jupyter notebook.**

**Operators -**

**Arithmetic Operators**

**Assignment Operators**

**Comparison Operators**

**Logical Operators**

**Identity Operators**

**Membership Operators**

**Operators**

They help us in performing operations

**Arithmetic Operators**

* **+** : addition
* **-** : subtraction
* [ \* ] : multiplication
* **/** : division
* **%** : modulus - gives remainder
* \*\* : Exponentiation
* **//** : Floor division - gives quoitent

**Data types**

* Integers
* float
* string
* boolean

**Intergers**

In [ ]:

1**+**1

Out[ ]:

2

In [ ]:

1 **\*** 3

Out[ ]:

3

In [ ]:

1 **/** 2

Out[ ]:

0.5

In [ ]:

2 **\*\*** 4

Out[ ]:

16

In [ ]:

4 **%** 2

Out[ ]:

0

In [ ]:

5 **%** 2

Out[ ]:

1

In [ ]:

5 **//** 2

Out[ ]:

2

In [ ]:

(2 **+** 3) **\*** (5 **+** 5)

Out[ ]:

50

In [ ]:

type(4)

*# int stands for integer*

Out[ ]:

int

**Float**

In [ ]:

type(1.25)

Out[ ]:

float

In [ ]:

1.25 **+** 8.0

Out[ ]:

9.25

In [ ]:

1 **+** 1.0

Out[ ]:

2.0

**Strings**

In [ ]:

'single quotes'

Out[ ]:

'single quotes'

In [ ]:

type('single quotes')

Out[ ]:

str

In [ ]:

"double quotes"

Out[ ]:

'double quotes'

In [ ]:

" wrap lot's of other quotes"

*# using single quotes within "" quotes*

Out[ ]:

" wrap lot's of other quotes"

In [ ]:

*# ' wrap lot's of other quotes'*

*# gives syntax error beacuse single quotes used inside of single quotes*

In [ ]:

*# escape character \*

' wrap lot\'s of other quotes'

**Booleans**

In [ ]:

**True**

Out[ ]:

True

In [ ]:

type(**True**)

Out[ ]:

bool

In [ ]:

**False**

Out[ ]:

False

**Variable Assignment**

In [ ]:

*# Can not start with number or special characters*

name\_of\_var **=** 2

In [ ]:

x **=** 2

y **=** 3

In [ ]:

x,y **=** 2,3 *# there should be equal no. on both sides also called unpacking*

In [ ]:

z **=** x **+** y

In [ ]:

z

Out[ ]:

5

In [ ]:

*# variable reassigned*

x **=** 10

x

Out[ ]:

10

**In-build functions**

python comes with many inbuild functions.

**Printing**

In [ ]:

x **=** 'hello'

In [ ]:

print(x)

hello

In [ ]:

print('Hi python, I am coming')

Hi python, I am coming

**Type**

In [ ]:

type(x)

Out[ ]:

str

In [ ]:

x **=** 8

type(x)

Out[ ]:

int

**Concatenation**

In [ ]:

print('Hi'**+**'There')

HiThere

In [ ]:

print('Hi','There')

Hi There

In [ ]:

print('Hi','There', 'hello')

Hi There hello

**Inter-converting string , float and int**

**Converting to int**

only the values which are number can be converted to int

In [ ]:

s **=** '1'

f **=** 10.0

In [ ]:

v **=** int(s)

fc **=** int(f)

print(v,fc)

1 10

In [ ]:

print(type(v),type(fc))

<class 'int'> <class 'int'>

In [ ]:

s **=** int('a')

**---------------------------------------------------------------------------**

**ValueError** Traceback (most recent call last)

**<ipython-input-13-ffb11c257ea4>** in <module>

**----> 1** s **=** int**('a')**

**ValueError**: invalid literal for int() with base 10: 'a'

**Converting to float**

only the values which are number can be converted to float

In [ ]:

print(float('1.0'))

print(float(10))

print(float('34'))

1.0

10.0

34.0

**Converting to string**

any value can be converted to float

In [ ]:

print(str(1.0))

print(str(10))

1.0

10

In [ ]:

print(type(str(1.0)))

print(type(str(10)))

<class 'str'>

<class 'str'>

**Operators**

**Assignment Operators**

**=** : assigns a value to a variable

**+=** : c += a is equivalent to c = c + a

**- =** : c -= a is equivalent to c = c - a

**(\*=)** : c *= a is equivalent to c = c*a

**/=** : c /= a is equivalent to c = c / a

**%=** : c %= a is equivalent to c = c % a

**//=** : c //= a is equivalent to c = c // a

In [ ]:

x **=** 1

y **=** 2

In [ ]:

x **+=** y *# x = x + y*

x

Out[ ]:

2

In [ ]:

x **-=** y

x

Out[ ]:

-2

In [ ]:

x **//=** y *# x = x // y*

x

Out[ ]:

-1

In [ ]:

x **\*=** y *# x = x \* y*

In [ ]:

x

Out[ ]:

-2

**Operators**

**Comparison Operators -**

They compares the values and returns bool if required

**==** : are equal

**!=** : not equal

**>** : greater than

**<** : samller than

**>=** : greater than or equal to

**<=** : smaller than or equal to

In [ ]:

1 **>** 2

Out[ ]:

False

In [ ]:

1 **<** 2

Out[ ]:

True

In [ ]:

1 **>=** 1

Out[ ]:

True

In [ ]:

1 **<=** 4

Out[ ]:

True

In [ ]:

1 **==** 1

Out[ ]:

True

In [ ]:

'hi' **==** 'bye'

Out[ ]:

False

In [ ]:

'hi' **!=** 'bye'

Out[ ]:

True

In [ ]:

'a' **==** 'A'

Out[ ]:

False

**Logic Operators**

**and** = requires both the statements to be true

**or** = requires any statement to be true

In [ ]:

**True** **and** **True**

Out[ ]:

True

In [ ]:

**True** **and** **False**

Out[ ]:

False

In [ ]:

**True** **or** **False**

Out[ ]:

True

In [ ]:

(1 **<** 2) **and** (2 **<** 3)

Out[ ]:

True

In [ ]:

(1 **>** 2) **and** (2 **<** 3) **and** (10 **<** 8) **and** (67 **==** 67)

Out[ ]:

False

In [ ]:

(1 **>** 2) **or** ('a' **<** 'A') **or** (10 **==** 10)

Out[ ]:

True

In [ ]:

(1 **==** 2) **or** (2 **==** 3)

Out[ ]:

False

**Comments**

They are used for giving a information about anything . They come in action when # i put before a line code

when the program runs seeing the # character the program skips that line.

In [ ]:

*# hi i want to tell you something*

In [ ]:

''' used for multi line comment also called doc string '''

**Data Structures**

* Lists
* Dictionaries
* Tuples
* Sets

**Lists**

Data type that stores any kind of data.

Are recognized by square brackets [ ]

In [ ]:

x **=** [1,2,3]

x

Out[ ]:

[1, 2, 3]

In [ ]:

type(x)

Out[ ]:

list

In [ ]:

['hi',1,1.25]

Out[ ]:

['hi', 1, 1.25]

In [ ]:

a **=** [[1,2],[45,25,43], 89 , 'hi' , 12.0]

a

Out[ ]:

[[1, 2], [45, 25, 43], 89, 'hi', 12.0]

**Tuples**

used for storing any data type. recoznized by small brackets ().

In [ ]:

a **=** (1,2,3)

a

Out[ ]:

(1, 2, 3)

In [ ]:

type(a)

Out[ ]:

tuple

In [ ]:

(44) *# It is not a tuple it is just 44 written in brackets.*

(44,) *# it is a tuple ,due to trailing comma after 44 within the brackets.*

Out[ ]:

(44,)

In [ ]:

('hih' , 1 ,1.0)

Out[ ]:

('hih', 1, 1.0)

In [ ]:

((1,2),[45,25,43],89,'ko')

Out[ ]:

((1, 2), [45, 25, 43], 89, 'ko')

**To convert and data type to tuple - use python tuple build in function**

In [ ]:

li **=** [1,2,3,4,5]

In [ ]:

tu **=** tuple(li)

In [ ]:

tu

Out[ ]:

(1, 2, 3, 4, 5)

**To convert and data type to list - use python build in function**

In [ ]:

list(tu)

Out[ ]:

[1, 2, 3, 4, 5]

**Indexing**

In [ ]:

s **=** 'abcde'

l **=** [34,28,35,43,'hi']

tu **=** (112.0 , 'k')

In [ ]:

s[4]

Out[ ]:

'e'

In [ ]:

l[3]

Out[ ]:

43

**Indexing with nested list and tuples.**

In [ ]:

tu **=** ((122,240),[45,25,43],89,'ko')

In [ ]:

tu[0]

Out[ ]:

(122, 240)

In [ ]:

tu[0][1]

Out[ ]:

240

**slicing**

sequence [start : end : step = 1 (optional)]

In [ ]:

lis **=** ['a' ,'b' ,'c' ,'d' ,'e' ,'f' ,'g' ,'h' ,'i' ,'j' ,'k']

In [ ]:

lis[1:8]

*# thing to note is that the element at 8th index is not included*

*# in simple terms the END is not included*

Out[ ]:

['b', 'c', 'd', 'e', 'f', 'g', 'h']

In [ ]:

lis[0:9]

Out[ ]:

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']

In [ ]:

x **=** lis[1:8]

x

Out[ ]:

['b', 'c', 'd', 'e', 'f', 'g', 'h']

In [ ]:

lis[2:6]

Out[ ]:

['c', 'd', 'e', 'f']

In [ ]:

lis[2:6:2]

Out[ ]:

['c', 'e']

In [ ]:

lis

Out[ ]:

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']

In [ ]:

lis[**-**3]

Out[ ]:

'i'

In [ ]:

lis[**-**6:]

*# index are always written from left to right*

Out[ ]:

['f', 'g', 'h', 'i', 'j', 'k']

In [ ]:

lis[**-**4:**-**1]

Out[ ]:

['h', 'i', 'j']

In [ ]:

x **=**lis[1:8:2]

x

Out[ ]:

['b', 'd', 'f', 'h']

**List methods**

In [ ]:

lis1 **=** [1,2,3,4,5]

lis2 **=** [5,4,3,2,1]

In [ ]:

lis1**.**append(10)

lis1

*# adds the element at last*

Out[ ]:

[1, 2, 3, 4, 5, 10]

In [ ]:

lis1 **\*** 3

In [ ]:

*# concatenating list*

lis1 **+** lis2

Out[ ]:

[1, 2, 3, 4, 5, 10, 5, 4, 3, 2, 1]

In [ ]:

*# changing the element by index*

lis1[3] **=** 'i am new element at index 3'

In [ ]:

lis1

Out[ ]:

[1, 2, 3, 'i am new element at index 3', 5, 10]

In [ ]:

*# l1.insert(index,element)*

lis1**.**insert(3,'a')

lis1

Out[ ]:

[1, 2, 3, 'a', 'i am new element at index 3', 5, 10]

In [ ]:

*# .remove(element) searches and removes the very first element not all*

lis1**.**remove('a')

lis1

Out[ ]:

[1, 2, 3, 'i am new element at index 3', 5, 10]

In [ ]:

*# count(element)*

lis1**.**count('i am new element at index 3')

Out[ ]:

1

In [ ]:

*# pop(index) , if no item is passed it removes element at -1 index*

lis1**.**pop(1)

Out[ ]:

2

In [ ]:

lis1

Out[ ]:

[1, 3, 'i am new element at index 3', 5, 10]

In [ ]:

lis1**.**remove('i am new element at index 3')

In [ ]:

*# reverses the list*

lis1**.**reverse()

lis1

Out[ ]:

[10, 5, 3, 1]

In [ ]:

lis1**.**sort(reverse**=True**)

lis1

Out[ ]:

[]

In [ ]:

*# .index(element) - gives the index of element*

lis1**.**index(3)

Out[ ]:

1

In [ ]:

*# clears the list*

lis1**.**clear()

lis1

Out[ ]:

[]

In [ ]:

c **=** lis1**.**copy()

In [ ]:

c

Out[ ]:

[]

**Tuple methods**

**The difference between tuples and list are that tuples are immutable (means they cannot be changed) where as list are mutable (means they can be changed)**

In [ ]:

tu **=** (1,2,'u')

In [ ]:

tu[0] **=** 'a'

*# this code can be applied on list because list are mutable*

**---------------------------------------------------------------------------**

**TypeError** Traceback (most recent call last)

**<ipython-input-142-c230822a761e>** in <module>

**----> 1** tu**[0]** **=** **'a'**

2 **# this code can be applied on list because list are mutable**

**TypeError**: 'tuple' object does not support item assignment

In [ ]:

tu**.**count(1)

Out[ ]:

1

In [ ]:

tu**.**index(2)

Out[ ]:

1

**Python identity operators**

They are used to compare objects , not if they are equal , but if they are actually the same object , eith same memory location.

* **is** - returns true if both objects are same object.
* **is not** - returns true if both objects are not he same

In [ ]:

'a' **is** 'a' *# ==*

Out[ ]:

True

In [ ]:

'a' **is** 'A'

Out[ ]:

False

In [ ]:

'a' **is** **not** 'A'

Out[ ]:

True

**Python membership operators**

They are used to test if the sequence is present in an obeject.

* **in** - returns true if a sequence with specified value is present in the object.
* **not in** - returns true if a sequence with specified value is not present in the object.

In [ ]:

s **=** 'hello man'

In [ ]:

'A' **in** s

Out[ ]:

False

In [ ]:

'he' **in** s

Out[ ]:

True

In [ ]:

'hjfff' **not** **in** s

Out[ ]:

True

**Dictionaries**

* Recognized by curly brackets { }
* They are mutable and can be nested like list and tuples
* They have a key and value pair , so they are also called associative pair
* A given key can oly appear once in a dictionary and both key and value can be any data type - str,int,float,tuple,list
* Each and value is seperated by a colon( : )
* Dictionary are unordered so indexing does not work on it. Instead keys are treated as index. whereas list and tuples are ordered
* when we convert a dictionary to list or a tuple by list() then only keys come in list and tuple not the values
* concatenation cannot be done on dictionary

In [ ]:

d **=** {'name':'sam' , 'age' : 23 , 'hair' : 'black' , 1 : 'id'}

*# key value key value*

In [ ]:

type(d)

Out[ ]:

dict

In [ ]:

d['name']

Out[ ]:

'sam'

In [ ]:

*# changing the value of a key*

d['name'] **=** 'rohit'

d

Out[ ]:

{'name': 'rohit', 'age': 23}

In [ ]:

*# adding a new and value*

d['place'] **=** 'Delhi'

d

Out[ ]:

{'name': 'rohit', 'age': 23, 'place': 'Delhi'}

**Dictionary methods**

In [ ]:

d**.**items() *# returns a view object that displays list of dictionary (key,value) in tuple pairs*

Out[ ]:

dict\_items([('name', 'rohit'), ('age', 23), ('place', 'Delhi')])

In [ ]:

d**.**keys() *# returns view object displaying a list of all keys*

Out[ ]:

dict\_keys(['name', 'age', 'place'])

In [ ]:

d**.**values()

In [ ]:

d**.**popitem() *# returns and removes any random item from dictionary*

print(d)

{'name': 'rohit'}

In [ ]:

*# d.pop(keyname,default\_value) - it removes and returns element from a dictionary having given key. ,*

*# default\_value is to be returned if key does not exist*

d**.**pop('nameeee' , 'no key')

Out[ ]:

'no key'

In [ ]:

*# d.update(iterable) - inserts the specified items to dictionary.*

d1 **=** {'country':'India' , 'seasons':6}

d**.**update(d1)

d

Out[ ]:

{'name': 'rohit', 'country': 'India', 'seasons': 6}

**Sets**

* It is a unordered collection of items, so elements cannot be called by index
* every item is unique (no duplicates)
* To convert a list , tuple, dictionary to the set use inbuild function set()

In [1]:

{1,2,3}

Out[1]:

{1, 2, 3}

In [2]:

{1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}

Out[2]:

{1, 2, 3}

**Sets methods**

In [3]:

s **=** {1,2,23,54,1,1,3,5,3,5,7,9,6,2,3,5,1,1,3,5,7,5,3,6,7}

s

Out[3]:

{1, 2, 3, 5, 6, 7, 9, 23, 54}

In [4]:

*# s.remove(element) removes the element*

s**.**remove(54)

s

Out[4]:

{1, 2, 3, 5, 6, 7, 9, 23}

In [5]:

*# s.add(element) - adds the element*

s**.**add(67)

s

Out[5]:

{1, 2, 3, 5, 6, 7, 9, 23, 67}

In [6]:

s1 **=** {1,2,3,4,5,6,7}

s2 **=** {5,2,6,89,65,2,34,64}

s1**.**difference(s2) *# s1 - s2 = only elements in s1 , that is minus all the elemts of s2 ,does not update s1 or s2 just provides the diff.*

Out[6]:

{1, 3, 4, 7}

In [7]:

s1**.**difference\_update(s2) *# updates the s1 inplace*

In [8]:

s1**.**intersection(s2) *# s1 and s2 ,s1 & s2 & s3 = s1.intersection(\*(any no. of sets) othersets) - returns only the common values of the set*

Out[8]:

set()

In [9]:

s1**.**intersection\_update(s2) *#updates s1*

In [10]:

s1**.**union(**\*** othersets) *# s1 or s2,s1 | s2 | s3- returns a set having all elements of all sets*

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

NameError Traceback (most recent call last)

<ipython-input-10-48e86966f9bb> in <module>()

----> 1 s1.union(\* othersets) # s1 or s2,s1 | s2 | s3- returns a set having all elements of all sets

NameError: name 'othersets' is not defined

In [ ]:

s1**.**update(s2) *#adds elements of s2 to s1 , works with list , tuple, dict , string*

**Python's In-Build Functions**

**Range**

* **Syntax** - range( start = 0(optional) , stop , step = 1 (optional) )
* It is a generator. means to see its output we use list() , tuple() , set() .
* It works only with Integers means it does not support Float. (To use Float numbers we can numpy arange method or we can create a function.)
* All three arguments can be postive ir negative.

In [ ]:

*# To show that range function is generator .*

print(range(10))

In [ ]:

list(range(20))

In [ ]:

**for** i **in** range(10):

print(i)

In [ ]:

range(10)[2]

In [ ]:

list(range(10,40,2))

**Length - Gives length of the iterable.**

**Syntax** - len(iterable)

In [ ]:

li **=** [1,2,3,4,5,6,7]

len(li)

**Max and MIn - Gives maximum and minimum value of the iterable**

**Syntax** - max( iterable ) , min( iterable )

In [ ]:

max(li)

In [ ]:

max(2,3,5,2,2,4,2,89)

In [ ]:

min(li)

In [ ]:

min(98,56,3,4,2,6,7)

**round()**

Returns the rounded off number of a floating number after the decimal points.

**syntax** - round( number , ndigits ) - ndigits - number of digits upto which we want the given number to be rounded off.

ndigits shold be smaller or equal to the number of digits after the decimal. if we put value of ndigit more ot will give same number without rounding it off.

In [ ]:

round(1.543)

In [ ]:

round(1.543 , 2)

**Sorted(iterable , key = None , reverse = False)**

Returns a list of iterable in ascending order.

works when all the elements in the iterable are of same data type.

In [ ]:

t **=** ('a','2','abc')

l **=** [9,8,3,8.5]

In [ ]:

sorted(l)

In [ ]:

sorted(t)

In [ ]:

ts **=** '1,2,3,4,\'a\''

sorted(ts)

**enumerate( iterable , start = 0(optional) )**

it generates a tuples of the element and its counter(can be said index if start not included) .To view should be called by list() ,for loop , tuple().

In [ ]:

g **=** [1,2,3,4,'a',8.0]

list(enumerate(g))

In [ ]:

list(enumerate(g ,10))

In [ ]:

dict(enumerate(g))

**help(object) - displays the info of object passed in it.**

In [ ]:

help(list)

**sum(iterable)**

Adds all the elements of the sequence.

In [ ]:

l **=** [1,2,3,4,8]

sum(l)

**reversed(sequence) -**

Return the reversed iterator of a given sequence. to view pass in list() ,tuple()

In [ ]:

l **=** [9,5,3,55,64,1,3]

reversed(l)

In [ ]:

list(reversed(l))

**zip()**

It takes iterables and aggregates them and returns an iterator of tuple means to have a view we have to pass it through list(), tuple().

It stops aggregates as soon as any sequence ends.

In [ ]:

l1 **=** ['a','g',1,5,9.8]

l2 **=** [9,8,'j',7,9,10]

list(zip(l1,l2))

In [ ]:

l1 **=** ['a','g',1,5,9.8]

l2 **=** [9,8,'j']

list(zip(l1,l2))

**.split(seperator(optional))**

It returns a list of strings after breaking the given string by the seperator.

In [ ]:

txt **=** "a v f v l 1 3 4"

In [ ]:

txt**.**split(' ')

In [ ]:

txt**.**split('3')

**.replace( old , new , count )**

replaces old values with new . works with strings only

In [ ]:

s **=** 'abc,abc,abc,abc,abc,abc,abc,abc'

s**.**replace('ab','A')

**While Loops**

It repeatedly executes the statement until it is True.

**Syntax** while expression: statement(s)

They can make infinite loops. to break out of a infinite loop use ctrl+c

In [ ]:

*# infinite loop*

i **=** 1

**while** i **<** 5:

print('This is a infinite loop')

**break**

This is a infinite loop

In [ ]:

i **=** 1

**while** i **<** 5:

print('{} is value of i'**.**format(i))

i **+=** 1

1 is value of i

2 is value of i

3 is value of i

4 is value of i

**For Loops**

It itereates over a sequence

In [ ]:

seq **=** [1,2,3,4,5]

In [ ]:

**for** item **in** seq:

print(item)

1

2

3

4

5

In [ ]:

**for** item **in** seq:

print('Yep')

Yep

Yep

Yep

Yep

Yep

In [ ]:

**for** jelly **in** seq:

print(jelly**+**jelly)

2

4

6

8

10

**if, else Statements**

Runs **if** statement and if the statement fails than **else** statement is run.

if can be alone but else follows a if or while statement.

In [ ]:

**if** 1 **<** 2:

print('Yep!')

Yep!

In [ ]:

**if** 1 **<** 2:

print('first')

**else**:

print('last')

first

In [ ]:

**if** 1 **>** 2:

print('first')

**else**:

print('last')

last

**In case of multiple if statements are used than all the if statements will run and the else follows the last if statement and there can be only one else statement.**

In [ ]:

x,y **=** 10,9

**if** x**>**y:

print('x is greater than y')

**if** x **>** 4:

print ('x is greater than 4')

**else**:

print('else statement ran')

x is greater than y

x is greater than 4

**if ,elif and else statements**

* elif = else if
* we use **elif** with **if** statement they can not be used independently.
* if there is **if** and **elif** statements in the program then , if and elif statement breaks the code if they are true.

In [ ]:

**if** 1 **==** 2:

print('first')

**elif** 3 **==** 3:

print('middle')

**else**:

print('Last')

middle

In [ ]:

x,y **=** 10,8

**if** x **>** y:

print('x is greater than y')

**elif** x **>** 5:

print('x is greater than 5')

**elif** x **>** 8:

print('x is greater than 8')

**else**:

print('else ran')

x is greater than y

In [ ]:

*# running the code by replacing elif with if*

x,y **=** 10,8

**if** x **>** y:

print('x is greater than y')

**if** x **>** 5:

print('x is greater than 5')

**if** x **>** 8:

print('x is greater than 8')

**else**:

print('else ran')

x is greater than y

x is greater than 5

x is greater than 8

**Python Functions**

* They are block of codes which run when called upon , they are made to decrease our work for writing the code again and again.
* They start with def which stands for define.
* They have small paranthesis with them followed by colon (:) which gives indentation to next line.
* Indentation is necsaary to make a particular code belong to a function
* They are called by using function name with small brackets and with parameters if defined.

In [ ]:

*# Function without parameters*

**def** myfunction():

print('hi')

In [ ]:

myfunction()

hi

In [ ]:

*# Function with parameters*

**def** addfunction(a,b):

print(a**+**b)

In [ ]:

addfunction(2,3)

5

In [ ]:

*# calling one fuction inside other*

**def** subtraction\_function(a,b):

print(a**-**b)

myfunction()

In [ ]:

subtraction\_function(10,9)

1

hi

In [ ]:

*# specifying the parameter default value.*

**def** add(a,b **=**10):

print(a**+**b)

add(5)

15

In [ ]:

*# we can update the defined value by passing new value.*

add(10,35)

45

**return**

A return statement causes our function to exit and hand back a value to the caller.

As soon as return is encountered , function exits without running the code written below return in function.

In [ ]:

**def** function():

print('in function')

**return** 1

x **=** function()

print('function returned' , str(x))

in function

function returned 1

In [ ]:

*# return can be used like = - return 1+1 , return a,b , return n > 1(will return boolean value)*

**pass**

If nothing is supoosed to happen in a code block , a pass is needed for such a block to not produce error.

In [ ]:

listA **=** [18, 21, 19, 29, 46]

**for** val **in** listA:

**pass**

*# if pass would not be written it would have thrown an error.*

**break**

It breaks the loop and comes out of the loop.

In [ ]:

**for** i **in** [10,2,3,4,5,6]:

print(i)

**break**

10

**continue**

The continue is preceeded by a condition . if the condition is True than continue will not run the code below and will move back starting step of the loop for processing next step.

In [ ]:

**for** letter **in** 'python':

**if** letter **==** 't':

**continue**

print(letter)

p

y

h

o

n

**Exception Handling**

**try ,except**

The python program terminates as soon as it encounters error.In python it can be syntax error or n exception. syntax error occurs when there is incorrect syntax like incomplete brackets , extra brackets ,wrong spelling

**try :**

run this code

**except:**

execute this code when there is a exception

In [ ]:

print(5**/**0)

**---------------------------------------------------------------------------**

**ZeroDivisionError** Traceback (most recent call last)

**<ipython-input-134-fad870a50e27>** in <module>

**----> 1** print**(5/0)**

**ZeroDivisionError**: division by zero

In [ ]:

**try**:

print(5**/**0)

**except**:

print('division by 0 not possibe')

division by 0 not possibe

In [ ]:

**try**:

print(5**/**5)

**except**:

print('hi')

1.0

In [ ]:

**try**:

print(5**/**5)

**except**:

print('hi')

1.0

In [ ]:

**try**:

print(5**/**0)

**except**:

print('hi')

hi

**Installing modules**

In standard distribution .

* In command prompt tye **pip install package**

In anaconda distribution

* In conda prompttype **conda install package**

**Difference between conda and pip installer**

pip installs the python package and dependencies required whether or not those conflict with other package previously installed .eg tensor flow can stop working when pip install a different version of numpy package

But conda does not let this happen , it figures out how t oinstall compatible dependencies.

**PYTHON INTRODUCTION AND BASIS SESSION ENDS**