###### Python Notes Nov. 29, 19

* General Purpose High Level Programming Language developed by **Guido Van Rossum** early 1989. first release of python version on Feb 20th 1991.
* Dynamic Programming Language no need to provide data type explicitly. according to data data-type automatic assign.
* variable data-type is not fixed it can change as data change.
* **Features:**

1. Simple
2. Freeware and Open-source
3. High level programming language
4. platform independent.
5. Portable
6. Dynamic Type
7. All oops concept
8. Interpreter (run time compilation)
9. Extensible ( we can add non python code to it)
10. embedded
11. Extensive Library

* **Limitation:**

1. Mobile application (lack of library support so difficult to develop mobile app)
2. Extensive Application ( Banking , Telephonic , where multiple service required)
3. Performance is Low ( run-time compilation and execution)

* **Different Type of Flavor of Python**

1. **CPython** C language application
2. **Jpython** java language application
3. **IronPython** c # sharp and .net language application
4. **RubyPython** ruby language to support large data
5. **AnacondaPython** python for Image Processing, Machine Learning , Data storage
6. **StacklessPython** multi-thread concurrency
7. **PyPy** python for Speed supported by JIT (just in time) compiler

* **Different Version of Python**

1. Python 2.0 (old language support may stop by python.org)
2. Python 3.0 (Independent Language) Latest Language will get support from Python.org

* **Variable / Identifier Name Rules**

1. a to z , A to Z, 0 to 9 and \_ allowed
2. name should not start with digit
3. keyword cant use
4. x normal, \_x protected , \_\_x private, \_\_x\_\_ magic

* **Only 33 keyword:**

True, False, none, and, or, not, is, if, else, elif, while, for, break, continue, return, in, yield, try, except, finally, raise, assert, import, from, as, class, det, puss, global, nonlocal, lamda, del with

switch and do-while not in python.

* **Data-Type:** Two main category **Fundamental Data-Type** and **Collection Related Data-Type**

**Fundamental:** int, float, bool, complex, str. it can hold only one value all are **mutable data-type.**

**Collection:** list, tuple, set, frozenset, dict, bytes, bytearray, range, all can hold multiple value.

\*long is available in python2 but not in Python3

1. **int:** whole number value. int holds value without decimal point. accept all Binaray(base2), Decimal(base10), Octal(base8), Hexadecimal(base16)
2. **float:** to represent decimal point value. it accept only Decimal(Base10)
3. **complex** to represent real and imaginary data a complex data eg: **10+20j**. 10 is real 20 is imaginary. real data accept all base. but imaginary data accept only Decimal(base10).
4. **bool** to represent bit value 1 and 0 with **True and False**.first char must be uppercase.
5. **str** sequence of character is string. single quote and double quote are allowed for single line string. and triple quote also for multiple line string. **index**ing can apply for **forward** direction also **Back-word** direction. s= “PRAFUL” # s[0]=’p’, and s[-1]= ‘L’
6. **list** collection of multiple value in order. it represent with square bracket **[ ]**. append() and remove() method can apply.
7. **tuple** read only version of list. it represent with parenthesis **( )**. append() and remove() method can not apply, it invoke error.
8. **set** collection of value irrespective of order. duplicate value not consider. represent with **{ }**. add() and remove method can applied.
9. **forzenset** read only set, everything as set type except it is mutable. we can not apply add() and remove() method.
10. **dict** to represent key:value pair. it represent with **{ }**
11. **range** to represent sequence of number in particular range. range(n) holds 0 to (n -1) value, range(s,e) holds s to e-1 value, range(s,e,I/d) value is between s and (e -1) and for increment positive value for decrements negative value.
12. **bytes** to represent binary data of (0-255) range. mutable variable.
13. **bytearray**  to represent binary data that is immutable.

* **Comparison between various python data-type**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Description** | **Mutable or Immutable** | **Order** | **Index / Slicing** | **Duplicate** |
| **int** | whole number that is without decimal point  a = 10 | Mutable | NA | NA | NA |
| **float** | floating numbers with decimal point  f = 4.8 | Mutable | NA | NA | NA |
| **bool** | bit represent  a = True  b = False | Mutable | NA | NA | NA |
| **complex** | to represent complex number that having **real and imaginary** data  c = 2 + 10j | Mutable | NA | NA | NA |
| **str** | sequence of character  s = “praful” | Mutable | NA | YES | NA |
| **list** | to represent collection of value in order  l = [10,20,30] | Mutable | YES | YES | YES |
| **tuple** | read only list  t = (10,20,30)  q=(40) int not tuple. q=(40,) is tuple | Immutable | YES | YES | YES |
| **set** | unorder collection of value  s = {10,20,30}  q=set() empty set | mutable | no | no | no |
| **forzenset** | same as set except immutable | immutable | no | no | no |
| **dict** | Dictionary is a pair of key:value  d={1 : ‘praful’} | mutable | no | yes | yes  key can not change |
| **byte** | group of byte (0-255) range | immutable | yes | yes | yes |
| **bytearray** | same as byte except it is mutable | mutable | yes | yes | yes |
| **range** | sequence of numbers | immutable | yes | yes | no |

* **none** data-type that does not contain any value.

def fun():

return 10;

def fun1():

print(“hello”)

x = fun() # x = 10

x = fun1() # x = none

* **escape character** special character having some meaning.

\n new-line

\t tab

\r carriage return

\b one space backward

\f form feed

\’ \” \\ to represent ‘ ,”, \, in string

# used for comment no multiple line comment, using single line comment in multiple line we can use as multiple line comment.

there is no **const** or **static** modifier in python.

* **slice operator [ begin : end ]**

slice is a part of existing thing. it having two argument in square bracket with column

[ : ] begin default is first index, end default is last index.

s = '0123456789'

print(s) # 0123456789

print(s[:]) # 0123456789

print(s[2:4]) # 23

print(s[5:]) # 56789

print(s[:3]) # 012

**output:**

0123456789

0123456789

23

56789

012

* **typecast** changing the type from another type

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Typecast** | **int** | **float** | **complex** | **bool** | **str** |
| **int()** | **-** | int(3.5) = 3 | **X** | int(True)=1  int(False)=0 | only base10 value  int(‘123’)=123 |
| **float()** | float(2) = 2.0 | **-** | **X** | float(True)=1.0  float(False)=0.0 | base10 value with decimal also  float(‘12.3’)=12.3 |
| **complex()** | complex(10) = 10 + 0j  complex(2,3) = 10 + 3j | same as int imaginary don’t accept float value | **-** | complex(True)=1+0j  complex(False)=0+0j | only one string  complex(‘4+3j’)=4+3j |
| **bool()** | bool(0)=False | bool(0.0)=False | bool(0+0j)=False | - | bool(‘’)=False |
| **str()** | str(123)=’123’ | str(2.3)=’2.3’ | str(1+4j)=’1+4j’ | str(True)=’True’  str(False)=’False’ | - |

* **Advanced Python 2-Dec-19**

**File Handling** permanently store data for future purpose

**text file** text data, name logs etc.

**binary file** image audio video etc.

* **methods of file-handling**

var = **open**(filename,mode)

filename = path of file

mode : for **text file**

|  |  |
| --- | --- |
| **MODE** | **Description** |
| **r**  **(default)** | read operation, open an existing file if file not available error will come **file not found**, file pointer **at begin.** |
| **w** | write operation, if specified file not available than new file will create. existing data will remove first then file pointer at **begin.** |
| **a** | append operation, open existing file and file pointer will point to **end of file.** data will not remove. if file not available it will create new file. |
| **r+** | read and write operation, first read than write file pointer at **begin** |
| **w+** | write and read operation, write than read file pointer at **begin.** |
| **a+** | append and read operation. file pointer at **end of file.** |
| **x** | exclusive . for write operation. if existing file is available **file is exist error.** always use to create file. |

\***for binary file** every mode ends with b**, rb, wb, ab, r+b, w+b, a+b, xb**

\***lowercase must**

open file with open method

f = **open(**‘filename’,’w’**)**

do required operation …

f.**close()**

close method should use to avoid **to many open file** error.

* **Various Property of file Object**

|  |  |
| --- | --- |
| f.name | name of file |
| f.mode | mode of file |
| f.closed | is file open |
| f.readable() | is read possible (return bool) |
| f.writable() | is write possible (return bool) |

**example:** open file and print property of file object.

f = open('demo.txt','w')

print('file name',f.name)

print('file mode',f.mode)

print('file is readable ?',f.readable())

print('file is writable ?',f.writable())

print('file is closed?"',f.closed)

f.close()

print("file is closed?",f.closed)

**output:**

file name demo.txt

file mode w

file is readable ? False

file is writable ? True

file is closed?" False

file is closed? True

* **Writing data to file**

|  |  |
| --- | --- |
| write(str) | write a string to file |
| writelines(group of line) | wirte lines to file |

**example:**

f = open('data.txt','w')

f.write("Praful Vanker\n")

l = ['praful','plot 25/2','sector 3new','gandhinagar','gujarat']

f.writelines(l)

f.close()

**output:** data.txt

Praful Vanker

prafulplot 25/2sector 3newgandhinagargujarat

* **Reading data from file**

|  |  |
| --- | --- |
| read() | read all data |
| read(n) | read n characters |
| readline() | read one line |
| readlines() | read all lines in list object |

**example:**

f = open('data.txt')

print(f)

file\_data = f.read()

print(file\_data)

f.close()

**output:**

<\_io.TextIOWrapper name='data.txt' mode='r' encoding='cp1252'>

Praful Vanker

prafulplot 25/2sector 3newgandhinagargujarat

**\*print(file\_data, end = ‘ ‘ )** print method wont add any \n to end.

**read line:**

f = open('data.txt')

lines = f.readlines()

print(lines)

for line in lines:

    print(line,end='')

**output:**

['PRAFUL\n', 'manilal\n', 'vanker\n']

PRAFUL

manilal

vanker

* **copy file example:**

f1 = open("input.txt")

f2 = open("output.txt",'w')

f2.write(f1.read())

f1.close()

f2.close()

* **with** use for automatic closing of file object.

with open('data.txt') as f:

    data = f.read()

    print(data)

    print(f.closed)

print(f.closed)

**output:**

PRAFUL

manilal

vanker

False

True

\* no need to close file. automatic file will close when with block completed.

* **Class and Object:**

class is a plan/blueprint/architect. and physical existence of class is object. so some method and data are decremented in same object is a class.

real world example:

if I want to buy iPhone\_8 so I went to shop 1 price is 45k, shop2 price is 44k, online price 43k. Banglore shop price 46k. so I conclude I should buy from online.

here iphone is same but price are different. so iphon is a like class. its specification and all is same in every shop.

**example:**

class Student:

    ''' this is just a demo of class '''

    def \_\_init\_\_(self,rollno,name):

        self.rollno = rollno

        self.name = name

    def talk(self):

        print("my name is ",self.name)

        print("my roll no is ",self.rollno)

s1 = Student(2154,'praful')

print(s1.name)

print(s1.rollno)

s1.talk()

s2 = Student(1954,'rakesh')

s2.talk()

**output:**

praful

2154

my name is praful

my roll no is 2154

my name is rakesh

my roll no is 1954

**3-Dec-19**

**Docc string:**

it represent the documentation of a class.

class Student:

‘’’ this is demo of docc string of class’’’

print(Student.\_\_docc\_\_)

help(Student)

**help(class name)** complete info of class with provided docc string.

* **self**

self is a reference variable which is **point to current object**. within python class to refer a python object we should use self variable.

1. every **instance method and constructor** first argument must be self
2. **PVM** python virtual machine is responsible to provide argument for self we are not require to provide it explicitly.
3. to declare and access instance variable **self** must use.
4. whatever first argument name. its treat as **self** only. so it’s recommended to use self only.

* **Constructor**

1. name of constructor is fixed **\_\_init\_\_().** for any class constructor name is \_\_init\_\_only.
2. whenever we create object constructor called automatic. explicitly we are not calling.
3. main objective of constructor is to declare and initialize instance variable.
4. for every object only one time called constructor method.
5. minimum one argument is require \_\_init\_\_(**self**):.

**Dec. 4, 19**

**instance variable:** for every object separate copy. for all object separate memory allocated. declare with self and inside constructor.

**static variable:** for every object only one copy. shared memory with all object.declare in class. it is also called class object.

1. inside constructor or any instance method by using **class-name** or **cls** variable.
2. inside static method by using class-name.
3. outside of class by using class-name.

real time example

if 100 student ==> pen, notebook. for each student they require pen and notebook (instance variable)

==> faculty (for every student only one faculty) (static variable)

class Student:

    ''' this is just a demo of class '''

    cname = "vvdn\_training"

    def**\_\_init\_\_**(self,rollno,name):

        self.rollno = rollno

        self.name = name

    def **talk**(self):

        print("my name is ",self.name)

        print("my roll no is ",self.rollno)

        print("cname = ",Student.cname)

s1 = Student(10,'praful')

s1.talk()

\*cname is static variable and rollno,name is instance variable.

**accessing static member:**

we can access static variable by using class name or object reference.

with in class by using class name, self, cls.

outside of class by using class name or object reference.

**example:**

class Student:

    a = 10

    def \_\_init\_\_(self):

        print('inside constructor')

        print(self.a)

        print(Student.a)

    def m1(self):

        print('inside instance method')

        print(self.a)

        print(Student.a)

    @classmethod

    def m2(cls):

        print('inside class method')

        print(cls.a)

        print(Student.a)

    @staticmethod

    def m3():

        print('inside static method')

        print(Student.a)

s1 = Student()

s1.m1()

s1.m2()

Student.m3()

print('outside class')

print(Student.a)

print(s1.a)

**output:**

inside constructor

10

10

inside instance method

10

10

inside class method

10

10

inside static method

10

outside class

10

10

**modify static variable:**

1. within class we should use class-name or cls variable.
2. outside of class we must use class-name.

**example:**

class Test:

    x = 10;

    def m1(self):

        self.x = 100

        self.y = 200

t1 = Test()

t2 = Test()

t3 = Test()

t1.x = 1000

t1.y = 2000

t3.m1()

print(t1.x,t1.y)

print(t3.x,t3.y)

print(t2.x,t2.y)

**output:**

1000 2000

100 200

Traceback (most recent call last):

File "c:/Users/pmvanker/Desktop/Dextop/python\_training\_vvdn/Program/classObject/static\_example.py", line 17, in <module>

print(t2.x,t2.y)

AttributeError: 'Test' object has no attribute 'y'

here t2 object is not yet having any instance variable.

* **Type of methods**

1. instance method: minimum one argument is **self** to access instance variable.
2. static method **@staticmethod:** general utility method. not require to object.
3. class method **@classmethod:** we can call class method by class-name. if we need to access only static variable. we are not required to create object of class. we can use **cls varibale**. @classmethod decorator compulsory we use. **we cant access instance variable.**

**@classmethod example:**

class Animal:

    legs = 4

    @classmethod

    def walk(cls,name):

        print('{} can walk with {} legs'.format(name,cls.legs))

Animal.walk('dog')

**output:**

dog can walk with 4 legs

\*without creating object we can access static variable of class.

* **setter and getter metod:**

alternative of constructor. we need to define it explicitly.

1. **Dec-19**

* **Inner Class:**

1. the class which is declare inside another class.
2. Without existing one type of object there is no chance of existing another type of object then should we go for inner class.

example:

class Outer:

    def \_\_init\_\_(self):

        print('Outer Constructor')

    class Inner:

        def \_\_init\_\_(self):

            print('Inner constructor')

o = Outer()

i = o.Inner()

\* here inner object i is possible only if outer object I is exist.

**output:**

Outer Constructor

Inner constructor

**real time example** . consider University that have many department. but all department exist because University. if there is no University then there is no need of its departments. so to get this real world approach inner class concept is require.

* **Garbage Collector**

gc module to determine garbage collector is enable disable

gc.isenabled(), gc.disable(), gc.enable()

**example:**

import gc

print(gc.isenabled())

gc.disable()

print(gc.isenabled())

gc.enable()

print(gc.isenabled())

**output:**

True

False

True

**example:**

import time

class Test:

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

        print('constructor')

    def \_\_del\_\_(self):#destructor

        print('distructor')

t = Test()

print('sleep.. 2sec before')

time.sleep(2)

t = None

print('end of programe')

**output:**

constructor

sleep.. 2sec before

distructor

end of programe

\*in if we implicitly assign None to object. then immediately Garbage collector will delete object.

otherwise when program last instruction is complete. before terminating program garbage collector delete all objects.

\***if all reference** variable pointing to object is none then and then it will delete. until it is not available to garbage collector.

**example:**

t1= Test()

t2=t1;

t3=t1;

t4=t1;

t1 = None

print('first object is none')

t2 = None

print('first object is none')

t3 = None

print('first object is none')

t4 = None

print('all object is none')

**output:**

constructor

first object is none

first object is none

first object is none

distructor

all object is none

* **Polymorphysm**

poly means many, morph means form , so one entity in multiple form is polymorphsm.

**Overloading:**

Operator Overloading

**Overriding:**

Method overriding

Constructor overriding

**+ operator:**

10 **+** 20 = 30

‘praful’ **+** ‘vanker’ = ‘prafulvanker’

\*here **+** operator in first case work as arithmetic plus operator and in second case concating **+** operator

if you want to use operator for custom data-type like class object . need to overload it.

operator overloading by magic method

**\_\_add\_\_()** method.

class book:

    def \_\_init\_\_(self,page):

        self.pages = page

    def \_\_add\_\_(self,other):

        return self.pages+other.pages

b1 = book(100)

b2 = book(200)

print(b1 + b2)

|  |  |
| --- | --- |
| **Operator** | **Magic Method** |
| + | \_\_add\_\_() |
| - | \_\_sub\_\_() |
| \* | \_\_mul\_\_() |
| / | \_\_div\_\_() |
| % | \_\_mod\_\_() |
| // | \_\_floordiv\_\_() |
| \*\* | \_\_pow\_\_() |
| += , -= , \*=, /=, %=, \*\*= | \_\_iadd\_\_(), \_\_isub\_\_(), \_\_imul()\_\_ … |
| > | \_\_gt\_\_() |
| >= | \_\_ge\_\_() |
| < | \_\_lt\_\_() |
| <= | \_\_le\_\_() |
| == | \_\_eq\_\_() |
| != | \_\_ne\_\_() |

by default print(object) **\_\_str\_\_()** internal method called .

def \_\_str\_\_(self):

        return str(self.pages)

print(b1)

now if we print(b1) is output of pages

add multiple object **example:**

class book:

    def \_\_init\_\_(self,page):

        self.pages = page

    def \_\_add\_\_(self,other):

        total = self.pages+other.pages

        return book(total)

    def \_\_str\_\_(self):

        return str(self.pages)

b1 = book(100)

b2 = book(200)

b3 = book(500)

print(b1 + b2 + b3)

**why python does not support method overloading?**

its very simple python is dynamic type. in c++ or java we can overload method by different argument type. but python does not support it.

|  |  |
| --- | --- |
| **C++/Java** | **Python** |
| fun(int)  fun(int, int)  fun(float)  fun(char) | fun( a)  fun(a ,b)  fun( c )  fun (d ) |

how we can determine in python fun function with arguments ?

That's why python does not support method overloading.

and there is not required in python.

fun function can accept any type and operate on it.

* **Variable Length Arguments.**

provide variable argument to method.

class Test:

    def sum(self,\*a):

        total = 0

        for x in a:

            total = total + x

        return total

t = Test()

print(t.sum(10,20))

print(t.sum(10,20,30))

print(t.sum())

* **Inheritance**

use the property of another class

class Parent:

    def house(self):

        print("house money 40000$")

    def land(self):

        print("land money 10000$")

class Child(Parent):

    def bike(self):

        print("bike money 5000$")

c = Child()

c.house()

c.land()

c.bike()

to call parent method in derived class call using **super().method\_name()**

* **Multi Threading / Tasking 11/12/2019**

Executing multiple task simultaneously

**Example: print the current thread name**

import threading

# threading module contain method current\_thread that return object and on that

# we call getName method

print('Thread:',threading.current\_thread().getName())

**output:**

Thread: MainThread

**Three Ways:**

1. Creating Threads without any class (functional way)
2. Creating Threads by extending Thread class
3. Creating Thread without extending Thread class

**\*note:** difference between

1. import threading
2. from threading import \*

here in 1st approach we need to type threading every time before calling any method of it.

but in 2nd approach we can directly call threading method.

**example: Two Thread with thread class()**

from threading import \*

def thread1():  # thread

    for i in range(10):

        print('thread 1')

t1 = Thread(target=thread1)# creating thread object

t1.start()  # starting thread

for i in range(10):

    print('Main Thread')

**output:**

thread 1

thread 1

Main Thread

thread 1

Main Thread

thread 1

Main Thread

thread 1

Main Thread

Main Thread

**\* Create Thread by extending Thread Class**

from threading import \*

class MyThread(Thread):

    def run(self):

        for i in range(5):

            print('thread name: ',current\_thread().getName())

t1 = MyThread()

t1.start()

for i in range(5):

    print("thread name:",current\_thread().getName())

**output:**

thread name: Thread-1

thread name: Thread-1

thread name: Thread-1

thread name: Thread-1

thread name: MainThread

thread name: Thread-1

thread name: MainThread

thread name: MainThread

thread name: MainThread

thread name: MainThread

\* **Create Thread without extending Thread class**

from threading import \*

class MyClass:

    def normal\_method(self):

        for i in range(5):

            print('thread:',current\_thread().getName());

obj = MyClass()

t1 = Thread(target=obj.normal\_method)

t1.start()

for i in range(5):

    print('thread:',current\_thread().getName());

**output:**

thread: Thread-1

thread: MainThread

thread: Thread-1

thread: MainThread

thread: Thread-1

thread: MainThread

thread: Thread-1

thread: MainThread

thread: MainThread

thread: Thread-1

* **Single Thread vs Multi Thread**

import time

from threading import \*

def duble(number):

    for n in number:

        time.sleep(1)

        print('duble :',n\*2)

def squre(number):

    for n in number:

        time.sleep(1)

        print('squre :',n\*n)

n = [1,2,3]

begin = time.time()

duble(n)

squre(n)

end = time.time()

print('time taken by single thread :',end - begin)

t1 = Thread(target=duble,args=(n,))

t2 = Thread(target=squre,args=(n,))

begin = time.time()

t1.start()

t2.start()

t1.join()

t2.join()

end = time.time()

print('time taken by multi thread :',end - begin)

**output:**

duble : 2

duble : 4

duble : 6

squre : 1

squre : 4

squre : 9

**time taken by single thread : 6.011902809143066**

duble : 2

squre : 1

duble : 4

squre : 4

duble : 6

squre : 9

**time taken by multi thread : 3.0180575847625732**

**note:** by using current\_thread().setName(‘mythread’) method we can change name of current thread

or current\_thread().name=’threadname’

**\*Thread identification:**

for every thread unic no is an identification number.

from threading import \*

def thread1():

    print('in thread1 id',current\_thread().ident)

t1 = Thread(target=thread1)

t1.start()

print('in main thread id',current\_thread().ident)

print('in main thread t1 id',t1.ident)

**output:**

in thread1 id 7208

in main thread id 8008

in main thread t1 id 7208

**\*Active Threads**

how many active thread currently running? to find it we can use **active\_count()** method

import time

from threading import \*

def mythread():

    print(current\_thread().getName())

    time.sleep(3)

    print(current\_thread().getName())

t1 = Thread(target=mythread,name='th1')

t2 = Thread(target=mythread,name='th2')

t3 = Thread(target=mythread,name='th3')

print('actvie threads :',active\_count())

t1.start()

print('t1 start now actvie threads :',active\_count())

t2.start()

print('t2 start now actvie threads :',active\_count())

t3.start()

print('t3 start now actvie threads :',active\_count())

t1.join()

t2.join()

t3.join()

print('all threads join now actvie threads :',active\_count())

**output:**

actvie threads : 1

th1

t1 start now actvie threads : 2

th2

t2 start now actvie threads : 3

th3

t3 start now actvie threads : 4

th1

th2

th3

all threads join now actvie threads : 1

**isAlive()** is method of thread object that return bool value.

it thread is alive True else False value return it can call with thread object **t1.isAlive()**

**join()** method to wait for thread to complete its task. so until all task is complete calling thread is in wait state. if in **main\_thread** we call **t1.join()** than main\_thread is in wait state until t1 thread complete its all task. after compete task join will done and main thread start execution next statements.

its use to synchronization the thread process. to resolve dependency.