Online certification course on Digital Image Processing using PYTHON

Introduction to PYTHON

Online certification course on Digital Image Processing using PYTHON.
Conducted by : Mrs. Usha Aniruddha Jogalekar
Assistant Professor, Department of Information Technology,
Pune Institute of Computer Technology, Pune

Data Types:
int
float
complex

Arithmetic Operators:

+
-
*
/ → returns the float quotient after division
// → returns integer quotient after division
% → returns remainder after division
**

a=10
b=3
a/b →3.3333333
a//b →3
a%b →1

If any one or both of a and b are float, then

a//b and a%b return float, with 0 after decimal point

Example:
a=10.0
b=3
a/b →3.3333333
a//b →3.0
a%b →1.0

```
a=5+4j
b=complex(2.4,7.8)
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

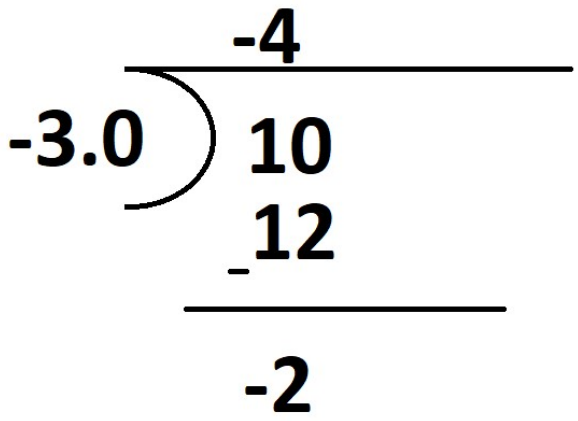
abs(x) → returns absolute value of x
divmod(x,y) → returns the pair (x//y,x%y)
If c is a complex number then

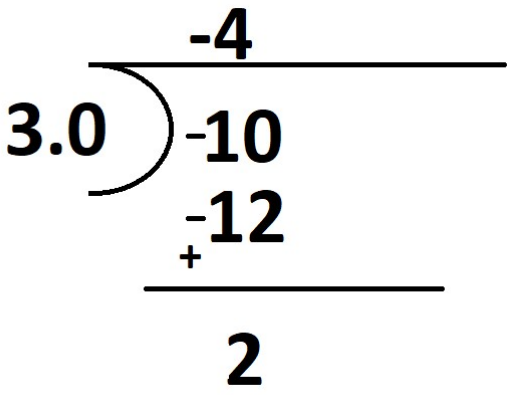
c.conjugate() → returns complex conjugate of c

c=1+2j

complex conjugate of c is 1-2j

```
a=10
b=-3.0
c=complex(a,b)
abs(b)→3.0
a**b → 0.001
pow(a,b)→0.001
c.conjugate()→ (10+3.0j)
divmod(a,b)→ (-4.0,-2.0)
divmod(-a,-b)→ (-4.0,2.0)
```


-4
-3.0
10
12
-2


-4
3.0
-10
-12
+
2

Bit wise operators

| → or
& → and
^ → exclusive or
~ → not
<< → shift left
>> → shift right

Relational operators

==
!=
>
>=
<
<=

Logical operators

and
or
not

x and y → y if both x and y are true, 0 otherwise
x or y → x if x is true, y if x is false, 0 otherwise
not x → true if x is false, false if x is true

```
a=5
b=10
c=a and b → 10
d=b and a → 5
e=(a>b) and b → False
f=(a<b) and b → 10
g=b and (a>b) → False
h=b and (a<b) → True
```

```
a=5
b=10
c=a or b → 5
d=b or a →10
e=(a>b) or b → 10
f=(a<b) or b → True
g=b or (a>b)→ 10
h=b or (a<b)→ 10
```

Control Structures:

if statement

if x :
  true part
else:
  false part

optional

**** x is any expression*

Nested if statement:

```
if x1:
  S1
else:
  if x2:
    S2
  else:
    S3
```

```
if x1:
  S1
elif x2:
  S2
else:
  S3
```

*** x1,x2 are expressions
S1, S2, S3 are set of instructions

Arrays :

Sequence of one or more values

```
a=[2,4,1.5,"PICT"]
print(a)
print(len(a))
```

Output:

```
[2, 4, 1.5, 'PICT']
4
```

```
a=[1,2,3,[4,5,6],[7,8],9]
print(a)
print(len(a))

Output:
[1, 2, 3, [4, 5, 6], [7, 8], 9]
6
```

```
Accessing single element in array
a=[1,2,3,[4,5,6],[7,8],9]
print(a[3])

Output:
 [4, 5, 6]
```

For Loop:

for x in R:
  do the needful

R can be:

range(N)
range(start,stop)
range(start,stop,step)
array

```
a=['a','b','xyz',123]
for x in a:
  print(x,end=' ')
print()
for x in range(5):
  print(x,end=' ')
print()
for x in range(-1,5):
  print(x,end=' ')
print()
for x in range(-1,3,2):
  print(x,end=' ')
```

➔

```
a b xyz 123
0 1 2 3 4
-1 0 1 2 3 4
-1 1
```

While Loop:

while expression:

.....

Write a program to:

1. Read a number and test it is prime or not
2. Find the maximum and minimum elements in the array using both for and while loops
3. Find n Fibonacci numbers
4. Solve a quadratic equation

Break and continue statements:

- Used in loops
- When break statement is executed, control moves out of the loop. i.e. to the next line of code after the loop
- When continue statement is executed, the remaining statements in the loop are skipped and control goes to the next iteration in the loop

Tuple

- Collection of objects separated by comma.
- A tuple is immutable
- The objects are
  - Ordered
  - Indexed – 0 to n-1
  - Unchangable
- Duplicates are allowed in a tuple
- len(name of the tuple) – used to find the elements in the tuple

Functions in PYTHON

- A function is a block of code, that is executed when it is called.
- Data can be passed to a function and a function can return data

- Creating a function – using ***def*** keyword
- Calling a function with its name

```
def xyz(): # function definition
    …… body of the function
_____________________________________________
xyz() # function call
```

Passing the data to a function

```
def xyz(parameters):
    …. Body of the function

_____________________________________

xyz(arguments)
```

The number of parameters and arguments must be same

```
def xyz():
  print("in the function xyz")
print("out side the function before the call")
xyz()
print("out side the function after the call")
```

```
def xyz(a,b):
  print("the values are ",a," and ",b)
xyz(1,2)
xyz(4,5)
```

```
def xyz(*par):
  print(type(par),end=' ')
  print(len(par))
xyz(1,2,3)
xyz(4,5)
```

```
def xyz(*par):
  print("\naccessing the elements using indexing")
  for i in range(len(par)):
    print(par[i],end=' ')
  print("\naccessing the tuple elements directly")
  for i in par:
    print(i,end=' ')
xyz(1,2,3,4,5)
xyz('PICT','SKNCOE','MMCOE','PVG')
xyz('PUNE','MUMBAI',12,32,68.9)
```

Function returning one or more values

- return v1,v2,…

- Assigning values to multiple variables

a,b,c=x,y,z

➔ a=x, b=y, c=z

- function xyz()
  - return r1,r2,r3,…,rn
- v1,v2,v3,…,vn=xyz()

- The function xyz() returns a tuple

```
def xyz(a):
  return a*a
print(xyz(2))
print(xyz(100))
```

```
def xyz(a):
  return 2*a, a*a, a*a*a
double,square,cube=xyz(5)
print(double,square,cube)
print(type(xyz(10)))
print(xyz(4))
```

Blank function

```
def xyz():
    pass
```

```
def xyz():
  pass
print("before call")
xyz()
print("after call")
```

Default arguments

```
def xyz(a,b=10)
    return a*b
```

Default arguments

```
def xyz(a,b=10)
   return a*b
```

xyz(5,3) ➔ 15

xyz(5) ➔ 50

Keyword arguments

- Using the parameters' names while calling the function
- Advantage: No need to remember the order of the parameters

```
def xyz(a,b):
  print("first argument is ",a," and second argument is ",b)
xyz(a=10,b=20)
xyz(b=20,a=10)
```

Recursion

- A function is said to be recursive, if it calls it self.
- Every recursive function should have a terminating condition.

Factorial of a number

$N! = N * (N-1)!$

Factorial of a number

N! =N * (N-1)!

Terminating condition :

0! = 1

```
def fact(n):
  if n<=1:
    return 1
  return n*fact(n-1)
print("factorial of 5 is ",fact(5))
```

Dictionary

- Similar to hash table, associative array
- Key-value pair
- Enclosed by {}

- a=[]
- a=()
- a={}

- <class 'list'>
- <class 'tuple'>
- <class 'dict'>

```
a=dict({1:5,'a':'b','abc':789,1:'x','abc':123})
```

- a={}
- a[1]='first'
- a['x']='second'
- a[3]='third'
- a['abc']='fourth'
- a[123]=5

- a={}
- a[1]='first'
- a['x']='second'
- a[3]='third'
- a['abc']='fourth'
- a[123]=5

- {1: 'first', 'x': 'second', 3: 'third', 'abc': 'fourth', 123: 5}

Variable length keyword arguments

```
def xyz(**par):
   ……
xyz(a=1,b=2…)
```

Docstring

- Describes the function in one string
- It is optional but considered as good practice
- Accessed using

  name of the function.__doc__

PYTHON Class

- A class has data (variables) and methods (functions)
- Access specifiers
  - Public
  - Protected _
  - Private __
- Every method should have first operator as ***self***
  - Equivalent to ***this*** operator in C++ or Java
- Constructor – method that executes when an object is created
  - __init__(self,…)

```
class a:
   x=None          #public
   _y=None         #protected
   __z=None        #private
   def __init__(self,x,y,z): # constructor
      self.x=x
      self._y=y
      self.__z=z
```

```
b=a(2,3,4)
print(b.x)
print(b._y)
print(b.__z) # Error - private variable
                    not accessible
```

```
# mthod in the class
def prnt__z(self):
    return self.__z


#call outside the class
print(b.prnt__z())
```

```
def pub_prod(self):
  return self.x*self._y*self.__z
def _prot_prod(self):
  return self.x*self._y*self.__z
def __priv_prod(self):
  return self.x*self._y*self.__z
```

```
print(b.pub_prod())
print(b._prot_prod())
print(b.__priv_prod()) # Error
```

Main function in Python

- Special variable __name__ = __main__
- Execution of a Python program starts from the first line at zero indentation.

Modules and packages

- Module : Collection of functions in one file
- Package : Directory containing different modules

NumPy

- Package used as a tool for scientific computing in Python

NumPy

Application include:

- Advanced array operations like add, multiply, slice, index
- Comprehensive mathematical functions
- Random number generation
- Linear algebra routines
- Fourier transforms

Matplotlib

- Data exploration and visualization library

Matplotlib

- Creation of graphs
  - Line plots
  - Bar charts
  - Pie charts

PIL - Pillow

- Python Image Library
- Pillow – package
- Image – open and save

NumPy

- Python C extension library for computing with arrays
- Efficient
- Image processing
- Signal processing
- Linear algebra

```
import numpy as np
a=np.array([2,5,1,7,6,9,3])
print(a)
```

- a=np.array([2,5,1,7,6.34,'asd',9,3])
- print(a)
- print(type(a)) ➔ <class 'numpy.ndarray'>

Changing the size of an array

- a=np.array([2,5,1,7,6.34,'asd',9,3,'qwe'])
- b=a.reshape(3,3)

- a=np.array([2,5,1,7,6.34,3.2,4,7,1,2,3,4])
- c=a.reshape(2,2,3)

Size and shape of an array

- a.shape – gives the dimensions of array and number of elements per dimension

Array operations

- c=a*2+3
- d=b*4+2

Array creation

- a=np.arange(10)
- b=np.linspace(0,1,5)
- c=np.linspace(0,10,5)
- d=np.linspace(0,10,7)

- [0 1 2 3 4 5 6 7 8 9]
- [0. 0.25 0.5 0.75 1.]
- [0. 2.5 5. 7.5 10.]
- [0. 1.66666667 3.33333333 5. 6.66666667 8.33333333 10.]

- a=np.zeros(10)
- b=np.ones(10)

- [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
- [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Blank array

- a=np.empty((5,4))

```
[[4.64660295e-310 4.64660449e-310 4.64660449e-310 4.64660449e-310]
 [4.64660449e-310 4.64660449e-310 4.64660449e-310 4.64660449e-310]
 [4.64660449e-310 4.64660449e-310 4.64660449e-310 4.64660449e-310]
 [4.64660449e-310 4.64660449e-310 4.64660449e-310 4.64660449e-310]
 [0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]]
```

Specifying the diagonals

- a=np.eye(4)

- [[1. 0. 0. 0.]
  [0. 1. 0. 0.]
  [0. 0. 1. 0.]
  [0. 0. 0. 1.]]

- a=np.diag([1,3,5,7,9])
- [[1 0 0 0 0] [0 3 0 0 0] [0 0 5 0 0] [0 0 0 7 0] [0 0 0 0 9]]

Indexing and slicing

- For a 2D array
- a[x:y,p:q] – indicates rows x to y-1 and columns p to q-1

- Write a program to read n and create n*n matrix. Create 4 matrices with four quadrants of the original matrix

```
import numpy as np
#read n from the user
print('enter n')
n=int(input())
a=np.arange(n*n)
b=a.reshape(n,n)
print(b)
p=n//2
c=b[0:p,0:p]
d=b[0:p,p:n]
e=b[p:n,0:p]
f=b[p:n,p:n]
print(c)
print(d)
print(e)
print(f)
```

Indexing and slicing

- For a 2D array with m rows and n columns
- a[x:y:z,p:q:r] – indicates rows x to y-1 with difference z
  and columns p to q-1 with difference r
- Default value of x and p = 0
- Default value of y=m and q =n
- Default value of z and r =1

NumPy array functions

- Concatenate(array1, array2, axis)
- For 1D array, axis=0
- For 2D array, if axis=0, array2 is concatenated as rows
  if axis=1, array 2 is concatenated as columns.

```
p=np.ones(5)
q=np.array([9,4,1,8])
r=np.concatenate([p,q])
a=np.zeros((5,4))
b=np.ones((5,3))
c=np.concatenate([a,b],axis=1)
d=np.zeros((2,3))
e=np.concatenate([b,d])
print(r)
print(c)
print(e)
```

```
[1. 1. 1. 1. 1. 9. 4. 1. 8.]

[[0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1.]
 [0. 0. 0. 0. 1. 1. 1.]]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Random number generator

- a=np.random.random(10) → 10 random numbers generated
- a=np.random.random((3,4)) → 12 random numbers generated
  - Range is 0 to 1
- rng=np.random.default_rng(***any number***)
- a=rng.integers(low=-50,high=100,size=10)
- a=rng.integers(low=50,high=100,size=[4,10])

Transpose of a matrix

- a=(np.arange(10)+1).reshape(5,2)
- print(a)
- b=a.T
- print(b)

- If a and b are lists : a+b → concatenation of a and b
- If a and b are ndarrays : a+b → addition of a and b

- Let a and b be ndarrays:
- a+b
- a-b
- a*b
- a>b
- np.dot(a,b) – matrix multiplication if a and b are 2D
- matmul is used for matrix multiplication for any dimesion

Python timer

import time

time.time()

time.perf_counter()

➔measures time in seconds from some unspecified time