DOC STRING ¶

```
Docstrings provide a convenient way of associating documentation with
functions. classes,
or modules.
```

In [1]:
```
def my_function(a,b):
    '''This function about adding two numbers'''
    return a+b
my_function(10,5)
```

Out[1]:
```
15
```

In [2]:
```
my_function.__doc__
```

Out[2]:
```
'This function about adding two numbers'
```

DATA TYPES

In [3]:
```
# Numeric
int_=10
float_=2.4
complex_=3+8j

# Sequence
list_=[1,2,3,4]
tuple_=(1,2,3,4)

#Mapping
dict_={'a':1,'b':2}

# set
set_={1,2,3,4}

#bool
bool_=True

# string
string_="Hello"
```

CONDITIONAL AND LOOP STATEMENT

In [4]:

```
# if .. else
if(True):
    print("hi")
else:
    print("hello")
# short hand   print("A") if a > b else print("B")
```

```
hi
```

In [5]:

```
# if ... elif ... else
flag=1
if(flag==0):
    print("if")
elif(flag==1):
    print("elif")
else:
    print("else")
```

```
elif
```

In [6]:

```
#Nested if
flag =1
if(flag>0):
    if(flag==1):
        print("inside if")
    else:
        print("inside else")
else:
    pass
```

```
inside if
```

In [7]:

```
# Looping
# for and while
i = 1
while i < 5:
  print(i)
  i += 1
else:
  print("i is no longer less than 5")
```

```
1
2
3
4
i is no longer less than 5
```

In [8]:

```
# break
i = 1
while i < 5:
    print(i)
    i += 1
    if(i==3):
        break

else:
  print("i is no longer less than 5")
```

```
1
2
```

In [9]:

```
# continue
# break
i = 1
while i < 5:
    print(i)
    i += 1
    if(i==3):
        continue

else:
  print("i is no longer less than 5")
```

```
1
2
3
4
i is no longer less than 5
```

In [10]:

```
# for Loop
for x in range(5):
  print(x, end=" ")
```

```
0 1 2 3 4
```

BOOLEAN

In [1]:

```
bool_1=bool(1)
bool_2=bool(0)
bool_3=bool(None)
print(bool_1,bool_2,bool_3)
```

```
True False False
```

CASTING

In [2]:

```
# int()
# str()
# bool()
# float()
# list()
# tuple()
# set()
```

STRING

In [3]:

```
str_="Hello World"
print(type(str))
isinstance(str_,str)
```

```
<class 'type'>
```

Out[3]:

```
True
```

In [4]:

```
str_[0]
```

Out[4]:

```
'H'
```

In [5]:

```
str_[-1]
```

Out[5]:

```
'd'
```

In [6]:

```
# str slicing
print(str_[2:5])  #forward
print(str_[-5:]) #backward
str_[::-1] #reverse
```

```
llo
World
```

Out[6]:

```
'dlroW olleH'
```

CONCATENATION

In [7]:

```
str_1="first"
str_2="second"
print(str_1+" "+str_2)
print(str_*3) #muliple
print(str_1,str_2,sep=":") #sep
print(str_1,str_2,end=" ,") #end
```

```
first second
Hello WorldHello WorldHello World
first:second
first second ,
In [8]:  # delete string
          del str_
          print(str_)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 3
      1 # delete string
      2 del str_
----> 3 print(str_)

NameError: name 'str_' is not defined
```

PARTITION

```
In [9]:  str_part="Hello, how are your?, are they?"
          print(str_part.partition("are")) #partition the sentence
```

```
('Hello, how ', 'are', ' your?, are they?')
```

```
In [10]: print(str_part.rpartition("are")) #last
```

```
('Hello, how are your?, ', 'are', ' they?')
```

FUNTIONS

```
In [11]: str_strip="**********     hi    *******"
          print(str_strip.strip("*"),end="")
```

```
     hi
```

```
In [12]: print(str_strip.rstrip("*"),end="")
```

```
**********     hi
```

```
In [13]: print(str_strip.lstrip("*"),end="")
```

```
     hi    *******
```

```
In [14]: print(str_part.count("are"))  #count
```

```
2
```

```
In [15]: str_ex="welcome everyone"
          print(str_ex.split())
```

```
['welcome', 'everyone']
```

In [16]:

```
str_ex.find("everyone")
```

Out[16]:

```
8
```

In [17]:

```
str_ex.replace("everyone",",hi")
```

Out[17]:

```
'welcome ,hi'
```

In [18]:

```
str_ex.index("welcome")
```

Out[18]:

```
0
```

In [19]:

```
num_="10"
print(num_.isnumeric(),
num_.isalnum(),
num_.isdecimal(),
num_.isdigit(),
num_.islower(),
num_.isupper(),
num_.isspace(),
num_.isascii())
```

```
True True True True False False False True
```

LIST

In [20]:

```
list_1=[1,2,3,4,5]
list_type=["hi",5.8,7,[5,7,8,9],(7,8,2)]
print(list_type[2])
type(list_type)
print(list_1[:5])
print(list_1[1:4])
print(list_1[-2:])
print(list_1[:-3])
```

```
7
[1, 2, 3, 4, 5]
[2, 3, 4]
[4, 5]
[1, 2]
```

LIST FUNCTION

In [21]:

```
print(list_1.append(10))
print(list_1.insert(0,"hi"))
print(list_1.pop())   #remove last element
print(list_1.pop(5))
del list_1 #.clear()
print(list_1)
```

```
None
None
10
5

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[21], line 6
      4 print(list_1.pop(5))
      5 del list_1 #.clear()
----> 6 print(list_1)

NameError: name 'list_1' is not defined
```

LOOPING & MEMBERSHIP

In [22]:

```
for i in list_type:
    print(i)
```

```
hi
5.8
7
[5, 7, 8, 9]
(7, 8, 2)
```

In [23]:

```
for i in enumerate(list_type):
    print(i)
```

```
(0, 'hi')
(1, 5.8)
(2, 7)
(3, [5, 7, 8, 9])
(4, (7, 8, 2))
```

In [24]:

```
# reverse & sort & sorted
sort_reverse=[3,6,4,57,8,2]
sort_reverse.sort()
print(sort_reverse)
print(sorted(sort_reverse))
sort_reverse.sort(reverse=True)
print(sort_reverse)
```

```
[2, 3, 4, 6, 8, 57]
[2, 3, 4, 6, 8, 57]
[57, 8, 6, 4, 3, 2]
```

In [25]:

```
any(sort_reverse)
```

Out[25]:

```
True
```

In [26]:

```
all(sort_reverse)
```

Out[26]:

```
True
```

TUPLE

In [36]:

```
(1,) #tuple
```

Out[36]:

```
(1,)
```

In [28]:

```
tuple_=(1,2.3,"hi",1)
print(tuple_.index(2.3))
tuple_.count(1)
```

```
1
```

Out[28]:

```
2
```

In [39]:

```
tuple_a=(1,4,6)
tuple_b=(7,4,8)
print(tuple_a+tuple_b)
tuple_x=tuple_a+tuple_b
```

```
(1, 4, 6, 7, 4, 8)
```

In [31]:

```
tuple_a*3
```

Out[31]:

```
(1, 4, 6, 1, 4, 6, 1, 4, 6)
```

In [33]:

```
for i in tuple_a:
    print(i)
```

```
1
4
6
```

In [35]:

```
# Asterisk
ex_tuple=(1,2,3,4,5,4,8,0)
(x,*y,z)=ex_tuple
print(x)
print(y)
print(z)
```

```
1
[2, 3, 4, 5, 4, 8]
0
```

In [40]:

```
tuple_x[:]
```

Out[40]:

```
(1, 4, 6, 7, 4, 8)
```

In [43]:

```
print(tuple_x[-5:-2])
print(tuple_x[1:3])
tuple_x[::-1]
```

```
(4, 6, 7)
(4, 6)
```

Out[43]:

```
(8, 4, 7, 6, 4, 1)
```

In [45]:

```
# tuple is immutable (unchangeable)
# one of the way to update
tuple_1=(88,)
tuple_x+=tuple_1
print(tuple_x)
```

```
(1, 4, 6, 7, 4, 8, 1, 2.3, 'hi', 1, 88)
```

SET

In [ ]:

```
# set is non duplicate mutable data type
```

In [47]:

```
set_={1,5,8,4,5,4} #duplicate are removed
set_
```

Out[47]:

```
{1, 4, 5, 8}
```

In [57]:

```
#function
set_a={1,8,5,7,2,6}
set_={1,5,8,4,5,4}
```

In [58]:

```
print(set_.add(5),
set_.difference(set_a),
set_.intersection(set_a),
set_.union(set_a),
set_.symmetric_difference(set_a),
set_.pop(),
set_.update([5,8,70]),
set_)
```

```
None {4} {8, 1, 5} {1, 2, 4, 5, 6, 7, 8} {2, 4, 6, 7} 8 None {1, 4, 5, 70,
8}
```

In [59]:

```
list(enumerate(set_))
```

Out[59]:

```
[(0, 1), (1, 4), (2, 5), (3, 70), (4, 8)]
```

DICT

```
In [60]: dict_={"A":1,"B":2,"C":3}
         dict_

Out[60]: {'A': 1, 'B': 2, 'C': 3}
```

```
In [61]: dict_.items()

Out[61]: dict_items([('A', 1), ('B', 2), ('C', 3)])
```

```
In [62]: dict_.values()

Out[62]: dict_values([1, 2, 3])
```

```
In [64]: dict_.keys()

Out[64]: dict_keys(['A', 'B', 'C'])
```

```
In [66]: a=[1,4,7,3]
         b={2,5,3,7}
         dict_.fromkeys(a,b)

Out[66]: {1: {2, 3, 5, 7}, 4: {2, 3, 5, 7}, 7: {2, 3, 5, 7}, 3: {2, 3, 5, 7}}
```

```
In [69]: # dict_.[]
         # dict_.get()           #Access the value
```

```
In [71]: dict_.pop("A")

Out[71]: 1
```

```
In [ ]: # (*)args & (**)kwargs
        # *args -> variable length Non Keyword Arguments (passed as a tuple)
        # **kwargs -> variable length Keyword Arguments (passed as a dictionary)
```

```
In [ ]:
```

LAMBDA , MAP , REDUCE , FILTER

```
In [2]: res = (lambda *args: sum(args))
        res(10,20) , res(10,20,30,40) ,  res(10,20,30,40,50,60,70)

Out[2]: (30, 100, 280)
```

```
In [4]: odd_num=[1,8,7,5,9,33]
        def twice(n):
            return n*2
        doubles = list(map(twice,odd_num))
        doubles

Out[4]: [2, 16, 14, 10, 18, 66]
```

In [5]:

```
from functools import reduce
def add(a,b):
    return a+b
sum_all = reduce(add,doubles)
sum_all
```

Out[5]:

```
126
```

In [6]:

```
list1 = [1,2,3,4,5,6,7,8,9]
def odd(n):
    if n%2 ==1: return True
    else: return False
odd_num = list(filter(odd,list1))
odd_num
```

Out[6]:

```
[1, 3, 5, 7, 9]
```

CLASS & OBJECT

In [7]:

```
class my_class:
    var_1 = 100
obj1 = my_class()
print(obj1.var_1)
```

```
100
```

In [10]:

```
class myclass:
    def __init__(self,a):
        self.var_1 = 100+a
obj1 = myclass(15)
print(obj1.var_1)
```

```
115
```

Inheritance

In [11]:

```python
# multi level , single , hierarchical inheritance
class person:
# Parent Class
    def __init__(self, name , age , gender):
        self.name = name
        self.age = age
        self.gender = gender
    def PersonInfo(self):
        print('Name :- {}'.format(self.name))
        print('Age :- {}'.format(self.age))
        print('Gender :- {}'.format(self.gender))

class employee(person): # Child Class
    def __init__(self,name,age,gender,empid,salary):
        person.__init__(self,name,age,gender)
        self.empid = empid
        self.salary = salary
    def employeeInfo(self):
        print('Employee ID :- {}'.format(self.empid))
        print('Salary :- {}'.format(self.salary))

class fulltime(employee): # Grand Child Class
    def __init__(self,name,age,gender,empid,salary,WorkExperience):
        employee.__init__(self,name,age,gender,empid,salary)
        self.WorkExperience = WorkExperience
    def FulltimeInfo(self):
        print('Work Experience :- {}'.format(self.WorkExperience))

class contractual(employee): # Grand Child Class
    def __init__(self,name,age,gender,empid,salary,ContractExpiry):
        employee.__init__(self,name,age,gender,empid,salary)
        self.ContractExpiry = ContractExpiry
    def ContractInfo(self):
        print('Contract Expiry :- {}'.format(self.ContractExpiry))
        print('Contractual Employee Details')
        print('****************************')

contract1 = contractual('Basit' , 36 , 'Male' , 456 , 80000,'21-12-2021')
contract1.PersonInfo()
contract1.employeeInfo()
contract1.ContractInfo()
print('\n \n')
```

```
Name :- Basit
Age :- 36
Gender :- Male
Employee ID :- 456
Salary :- 80000
Contract Expiry :- 21-12-2021
Contractual Employee Details
****************************
```

In [13]:

```
# Super Class
class Father:
    def __init__(self):
        self.fathername = str()
 # Super Class
class Mother:
    def __init__(self):
        self.mothername = str()
 # Sub Class
class Son(Father, Mother):
    name = str()
    def show(self):
        print('My Name :- ',self.name)
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
s1 = Son()
s1.name = 'Bill'
s1.fathername = "John"
s1.mothername = "Kristen"
s1.show()
```

```
My Name :-  Bill
Father : John
Mother : Kristen
```

In [1]:

```
class person:   # Parent Class
    def __init__(self, name , age , gender):
        self.name = name
        self.age = age
        self.gender = gender
    def PersonInfo(self):
        print('Name :- {}'.format(self.name))
        print('Age :- {}'.format(self.age))
        print('Gender :- {}'.format(self.gender))

class student(person): # Child Class
    def __init__(self,name,age,gender,studentid,fees):
        super().__init__(name,age,gender)
        self.studentid = studentid
        self.fees = fees
    def StudentInfo(self):
        super().PersonInfo()
        print('Student ID :- {}'.format(self.studentid))
        print('Fees :- {}'.format(self.fees))
stud = student('Asif' , 24 , 'Male' , 123 , 1200)
print('Student Details')
print('---------------')
stud.StudentInfo()
```

```
Student Details
---------------
Name :- Asif
Age :- 24
Gender :- Male
Student ID :- 123
Fees :- 1200
```

Iterator

In [ ]:

```
# iter()
# next() (StopIteration) -> .__next__()
```

In [14]:

```
m=[1,5,8,2,7,6]
x=iter(m)
print(x.__next__())
print(x.__next__())
print(x.__next__())
```

```
1
5
8
```

Decorator

In [15]:

```
def install_decorator(func):
    def wrapper():
        print("read terms and conditions")
        return func()
    return wrapper()

@install_decorator
def A_user():
    print("login")

@install_decorator
def B_user():
    print("login_page")
```

```
read terms and conditions
login
read terms and conditions
login_page
```