FUNCTIONS

- Defining a function

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

You can pass data, known as parameters, into a function.

A function can return data as a result.Calling a function

- Function Parameters

Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name.

- **Types of parameters**
- **Required arguments**

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

Example:

```
# Function definition is here
def printme( str ):
   #"This prints a passed string into this function"
   print(str)
   return
# Now you can call printme function
printme()     ===>Error
```

output:

```
Traceback (most recent call last):
   File "test.py", line 11, in <module>
      printme()
TypeError: printme() takes exactly 1 argument (0 given)
```

- **default parameters**

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.
Example:

```
# Function definition is here
def printinfo( name, age = 35 ):
  # "This prints a passed info into this function"
   print("Name: ", name)
   print( "Age ", age)
   return
# Now you can call printinfo function
printinfo( age=50, name="miki" )
printinfo( name="miki" )
```
output:
Name: miki
Age 50
Name: miki
Age 35

- **key word arguments**

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.
This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.
Example:

```
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print("Name: ", name)
   print("Age ", age)
   return
# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

Output:
Name: miki
Age 50

- **arbitrary arguments**

Sometimes, we do not know in advance the number of arguments that will be passed into a function.Python allows us to handle this kind of situation through function calls with arbitrary number of arguments.
In the function definition we use an asterisk (*) before the parameter name to denote this kind of argument.
Example:

```
def greet(*names):
   """This function greets all
   the person in the names tuple."""

   # names is a tuple with arguments
   for name in names:
      print("Hello",name)

greet("Monica","Luke","Steve","John")
```

Output:

```
Hello Monica
Hello Luke
Hello Steve
Hello John
```

- **Return statement in functions**

A return statement is used to end the execution of the function call and “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned.
Note: Return statement can not be used outside the function.
Syntax:

```
def fun():
    statements
    .
    .
    .
    return [expression]
```

Example:

```
def add(a, b):
    # returning sum of a and b
```

```
    return a + b

def is_true(a):

    # returning boolean of a
    return bool(a)

# calling function
res = add(2, 3)
print("Result of add function is {}".format(res))

res = is_true(2<5)
print("\nResult of is_true function is {}".format(res))
```

Output:

Result of add function is 5
Result of is_true function is True

- **Handling return values**
  - **Using Object:**

```
class Test:
    def __init__(self):
        self.str = "Hello"
        self.x = 20
def fun():
    return Test()

t = fun()
print(t.str)
print(t.x)
```

output:

```
Hello
 20
```

  - **Using Tuple:** A Tuple is a comma separated sequence of items. It is created with or without (). Tuples are immutable.

    Example:

```
def fun():
    str = "NAveen"
```

```
    x   = 20
    return str, x;
str, x = fun()
print(str)
print(x)
```

output:

```
NAveen
20
```

- **Using a list:** A list is like an array of items created using square brackets. They are different from arrays as they can contain items of different types. Lists are different from tuples as they are mutable.

  Example:

```
def fun():
    str = "NAveen"
    x   = 20
    return [str, x];
list = fun()
print(list)
```

output:

```
['NAveen', 20]
```

- **Using a Dictionary:** A Dictionary is similar to hash or map in other languages

  Example:

```
def fun():
    d = dict();
    d['str'] = "Welcome"
    d['x']   = 20
    return d
d = fun()
print(d)
```

output:

```
{'str': 'Welcome', 'x': 20}
```

Print(), Type()and Id()Functions:

id() function : id() is an inbuilt function in python

syntax: id(object)

As we can see the function accepts a single parameter and is used to return the identity of an object. This identity has to be unique and constant for this object during the lifetime.

for ecample,

```
>>> id(1025)
1603982556432
>>> id("naveen")
1603982428464
```

Type() function:

type() method returns class type of the argument(object) passed as parameter. type() function is mostly used for debugging purposes.

Two different types of arguments can be passed to type() function, single and three argument. If single argument type(obj) is passed, it returns the type of given object. If three arguments type(name, bases, dict) is passed, it returns a new type object.

syntax:

i) type(object**)**

ii) type(name, bases, dict)

Parameters :

name : name of class, which later corresponds to the __name__ attribute of the class.

bases : tuple of classes from which the current class derives. Later corresponds to the __bases__ attribute.

dict : a dictionary that holds the namespaces for the class. Later corresponds to the __dict__ attribute.

Returntype :

returns a new type class or essentially a metaclass.

examples:

```
i) print(type([]) is list)                  #True

ii) print(type([]) is not list)     #False

iiiprint(type(()) is tuple)                 #True

print(type({}) is dict)             #True

print(type({}) is not list)         #True
```

print() function:

The print() function prints the given object to the standard output device (screen) or to the text stream file.

The full syntax of print() is:

print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)

print() Parameters

objects - object to the printed. * indicates that there may be more than one object

sep - objects are separated by sep. Default value: ' '

end - end is printed at last

file - must be an object with write(string) method. If omitted it, sys.stdout will be used which prints objects on the screen.

flush - If True, the stream is forcibly flushed. Default value: False

Example:

```
print("Python is fun.")
a = 5
# Two objects are passed
print("a =", a)
b = a
# Three objects are passed
print('a =', a, '= b')
```

output:

```
Python is fun.
a = 5
a = 5 = b
```

Input()and raw input()functions

input() function:

Python input() function is used to take the values from the user. This function is called to tell the program to stop and wait for the user to input the values. It is a built-in function.

The input() function is used in both the version of Python 2.x and Python 3.x. In Python 3.x, the input function explicitly converts the input you give to type string. But Python 2.x input function takes the value and type of the input you enter as it is without modifying the type.

Example:

```
val1 = input("Enter the name: ")
# print the type of input value
print(type(val1))
print(val1)
```

```
val2 = input("Enter the number: ")
print(type(val2))

val2 = int(val2)
print(type(val2))
print(val2)
```

output:

```
Enter the name: naveen
<class 'str'>
naveen
Enter the number: 11421
<class 'str'>
<class 'int'>
11421
```

raw_input() function:

Python raw_input function is used to get the values from the user. We call this function to tell the program to stop and wait for the user to input the values.

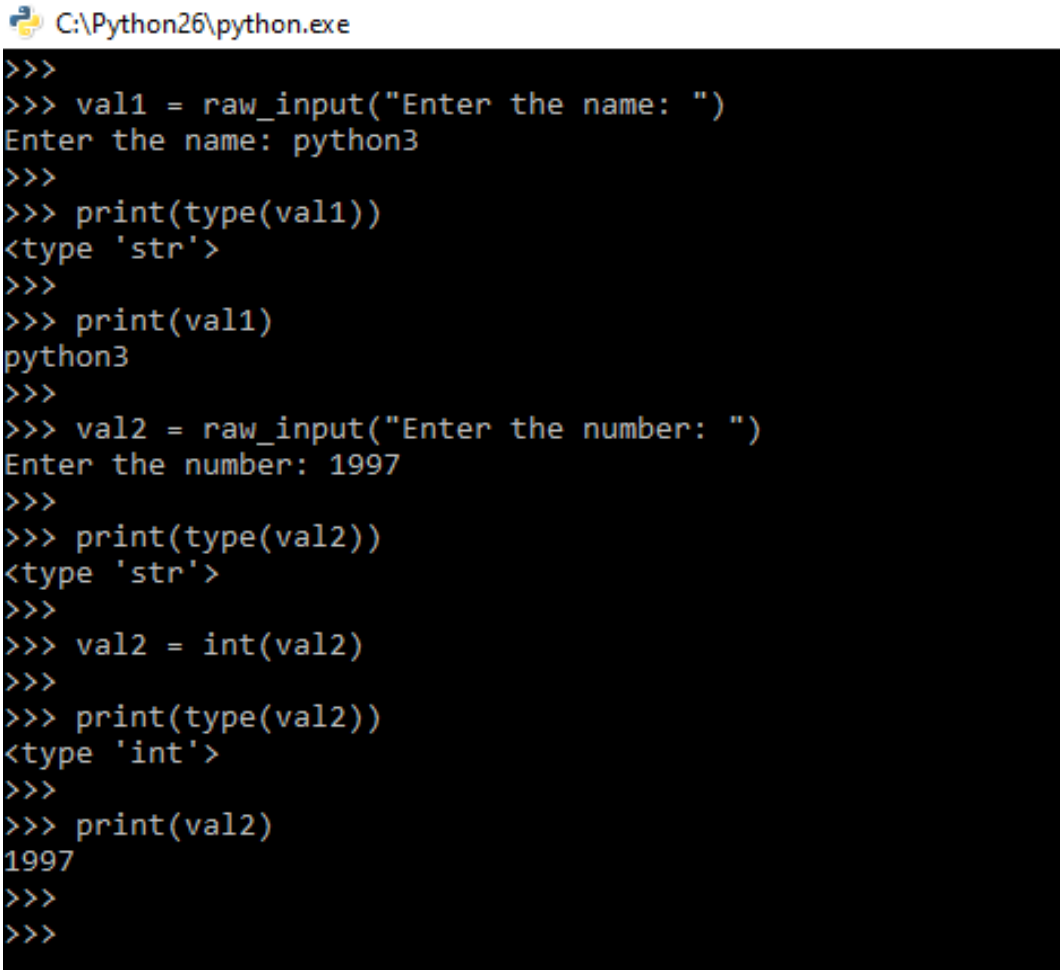
It is a built-in function. The input function is used only in Python 2.x version. The Python 2.x has two functions to take the value from the user. The first one is input function and another one is raw_input() function. The raw_input() function is similar to input() function in Python 3.x. Developers are recommended to use raw_input function in Python 2.x.

Example:

```
val1 = raw_input("Enter the name: ")
print(type(val1))
print(val1)

val2 = raw_input("Enter the number: ")
print(type(val2))
val2 = int(val2)
print(type(val2))
print(val2)
```

output:



Type Conversion functions

Python defines type conversion functions to directly convert one data type to another.

1. int(a,base) : This function converts any data type to integer. 'Base' specifies the base in which string is if data type is string.

example: s = "10010"

printing string converting to int base 2

c = int(s,2)

print ("After converting to integer base 2 : ", end="")

print (c)

output:

After converting to integer base 2 : 18

2. float() : This function is used to convert any data type to a floating point number.

example:s = "10010"

printing string converting to float

e = float(s)

print ("After converting to float : ", end="")

print (e)

output:

After converting to float : 10010.0

3.ord() : This function is used to convert a character to integer.

s = '4'

printing character converting to integer

c = ord(s)

print ("After converting character to integer : ",end="")

print (c)

output:

After converting character to integer : 52

4. hex() : This function is to convert integer to hexadecimal string.

printing integer converting to hexadecimal string

c = hex(56)

print ("After converting 56 to hexadecimal string : ",end="")

print (c)

output:

After converting 56 to hexadecimal string : 0x38

5. oct() : This function is to convert integer to octal string.

```
# printing integer converting to octal string
c = oct(56)
print ("After converting 56 to octal string : ",end="")
print (c)
```

output:

```
After converting 56 to octal string : 0o70
```

6. tuple() : This function is used to convert to a tuple.

7. set() : This function returns the type after converting to set.

8. list() : This function is used to convert any data type to a list type.

9. dict(): This function is used to convert a tuple of order (key,value) into a dictionary.

example:

```
# initializing tuple
tup = (('a', 1) ,('f', 2), ('g', 3))
# printing tuple converting to expression dictionary
c = dict(tup)
print ("After converting tuple to dictionary : ",end="")
print (c)
```

output:

```
After converting tuple to dictionary : {'a': 1, 'f': 2, 'g': 3}
```

10. str(): Used to convert integer into a string.

```
a = 1
b = 2
# printing integer converting to string
c = str(a)
```

print ("After converting integer to string : ",end="")

print (c)

output:

After converting integer to string : 1

11. complex(real, imag): This function converts real numbers to complex(real,imag) number.

example:

c = complex(1,2)

print ("After converting integer to complex number : ",end="")

print (c)

output:

After converting integer to complex number : (1+2j)