PYTHON FUNCTIONS

Call by reference in Python

Example 1 Passing Immutable Object (List)

```
#defining the function
def change_list(list1):
    list1.append(20);
    list1.append(30);
print("list inside function = ",list1)

#defining the list
list1 = [10,30,40,50]

#calling the function
change_list(list1);
print("list outside function = ",list1);
```

Output:

```
list inside function = [10, 30, 40, 50, 20, 30]
list outside function = [10, 30, 40, 50, 20, 30]
```

Example 2 Passing Mutable Object (String)

```
#defining the function
def change_string (str):
    str = str + " Hows you";
    print("printing the string inside function :",str);

string1 = "Hi I am there"

#calling the function
change_string(string1)

print("printing the string outside function :",string1)
```

Output:

```
printing the string inside function : Hi I am there Hows you
print ing the string outside function : Hi I am there
```

Types of arguments

- Required arguments

Example 1

```
#the argument name is the required argument to the function func
def func(name):
    message = "Hi "+name;
    return message;
name = input("Enter the name?")
print(func(name))
```

Output:

```
Enter the name?John Hi John
```

Example 2

- #the function simple_interest accepts three arguments and returns the simple interest accordingly
- **def** simple_interest(p,t,r):
-     **return** (p*t*r)/100
- p = float(input("Enter the principle amount? "))
- r = float(input("Enter the rate of interest? "))
- t = float(input("Enter the time in years? "))
- **print**("Simple Interest: ",simple_interest(p,r,t))
- **Output:**
- Enter the principle amount? 10000 Enter the rate of interest? 5 Enter the time in years? 2 Simple Interest: 1000.0

Example 3

- #the function calculate returns the sum of two arguments a and b
- **def** calculate(a,b):
-     **return** a+b
- calculate(10) # this causes an error as we are missing a required arguments b.
- **Output:**
- TypeError: calculate() missing 1 required positional argument: 'b'

Keyword arguments

```
#function func is called with the name and
 message as the keyword arguments
def func(name,message):
    print("printing the message with",name,"and
    ",message)
func(name = "John",message="hello") #name an
    d message is copied with the values John and
    hello respectively
```

Output:

```
printing the message with John and hello
```

Example 2 providing the values in different order at the calling

- #The function simple_interest(p, t, r) is called wit h the keyword arguments the order of arguments doesn't matter in this case
- **def** simple_interest(p,t,r):
-     **return** (p*t*r)/100
- **print**("Simple Interest: ",simple_interest(t=10,r=10,p=1900))
- **Output:**
- Simple Interest: 1900.0

Example 3

#The function simple_interest(p, t, r) is called with the keyword arguments.

```
def simple_interest(p,t,r):
    return (p*t*r)/100


print("Simple Interest: ",simple_interest(time=10,rate=10
    ,principle=1900)) # doesn't find the exact match of the
    name of the arguments (keywords)
```

Output:

```
TypeError: simple_interest() got an unexpected keyword
    argument 'time'
```

mix of the required arguments and keyword arguments

- the required argument must not be given after the keyword argument, i.e., once the keyword argument is encountered in the function call, the following arguments must also be the keyword arguments.

- Example 4
- **def** func(name1,message,name2):
-     **print**("printing the message with",name1,",",message,",and",name2)

- func("John",message="hello",name2="David") #the first argument is n ot the keyword argument
- **Output:**
- printing the message with John , hello ,and David
- The following example will cause an error due to an in-proper mix of keyword and required arguments being passed in the function call.
- Example 5
- **def** func(name1,message,name2):
-     **print**("printing the message with",name1,",",message,",and",name2)

- func("John",message="hello","David")
- **Output:**
- SyntaxError: positional argument follows keyword argument

Default Arguments

- Example 1
- **def** printme(name,age=22):
-     **print**("My name is",name,"and age is",age)
- printme(name = "john") #the variable age is n ot passed into the function however the defau lt value of age is considered in the function
- **Output:**
- My name is john and age is 22

- Example 2
- **def** printme(name,age=22):
-     **print**("My name is",name,"and age is",age)
- printme(name = "john") #the variable age is not passed into the function however the default value of age is considered in the function
- printme(age = 10,name="David") #the value of age is overwritten here, 10 will be printed as age
- **Output:**
- My name is john and age is 22 My name is David and age is 10

Variable length Arguments

- Example
- **def** printme(*names):
-     **print**("type of passed argument is ",type(names))
-     **print**("printing the passed arguments...")
-     **for** name **in** names:
-         **print**(name)
- printme("john","David","smith","nick")
- **Output:**
- type of passed argument is <class 'tuple'> printing the passed arguments... john David smith nick

Scope of variables

depend upon the location where the variable is being declared

two types of scopes

- Global variables
- Local variables

Example 1

```
def print_message():
    message = "hello !! I am going to print a message." # the variable message is local to the function itself
    print(message)
print_message()
print(message) # this will cause an error since a local variable cannot be accessible here.
```

Output:

```
hello !! I am going to print a message. File "/root/PycharmProjects/PythonTest/Test1.py", line 5, in print(message) NameError: name 'message' is not defined
```

- Example 2

```
def calculate(*args):
    sum=0
    for arg in args:
        sum = sum +arg
    print("The sum is",sum)
sum=0
calculate(10,20,30) #60 will be printed as the sum
print("Value of sum outside the function:",sum) # 0 will be printed
```

- **Output:**

```
The sum is 60 Value of sum outside the function: 0
```