PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Functions: Variable number of positional args and Keyword args

Prof. Sindhu R Pai
PCPS Theory Anchor - 2024
Department of Computer Science and Engineering

Function Call: Arguments and Parameters

- In the function definition, variables are specified which receive the arguments when the function is called or invoked. **These are called parameters**.
- The parameters are **always variables**.
- When the function call is made, the control is transferred to the function definition. The arguments are evaluated and copied to the corresponding parameters.
- This is called **parameter passing by value or call by value**.
- The value of arguments are passed by value to the parameters.

Function Call: Arguments and Parameters

- When a function performs a specified task, it may require some values to operate on.
- These values have to be provided when calling a function as input.
- These values have to be put within the pair of round parentheses in the function call.
- **They are called as arguments**.

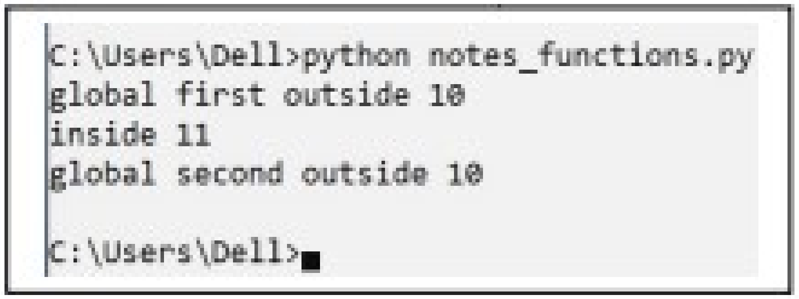
Function: Global and Local variables

There are two types of variables

- **Global Variables:** Variables created outside all functions. Scope of Global variable is within the file and outside that file too.
- **Local variables:** Variables created inside the function and accessible only to that function. Scope of the local variable is within the function.

Example 1:

```
x=10
print("global first outside",x)
def f1():
     x=11
     #local variable x.
    #Life and scope is only within this function
    print("inside",x)
f1()
print("global second outside",x)
```



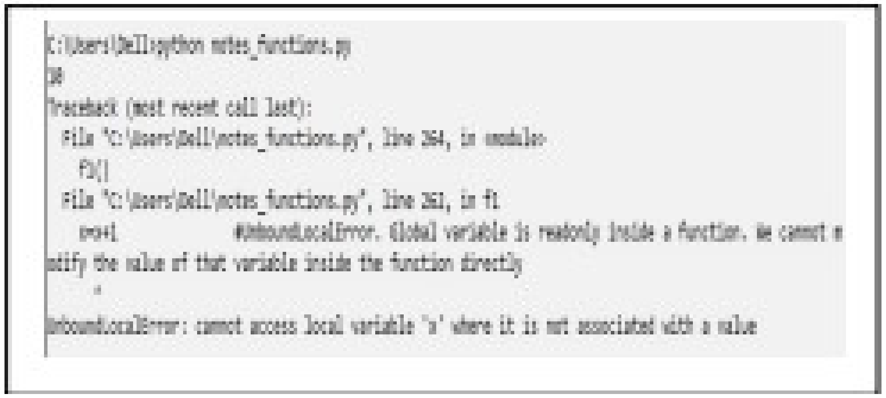
```
C:\Users\Dell>python notes_functions.py
global first outside 10
inside 11
global second outside 10

C:\Users\Dell>
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Function: Global and Local variables

Example 2: Global variable is read-only inside a function. We cannot modify the value of that variable inside the function directly.

```
x=10
print(x)
def f1():
    x=x+1 #UnboundLocalError.
    print("inside",x)
f1()
print("outside",x)
```



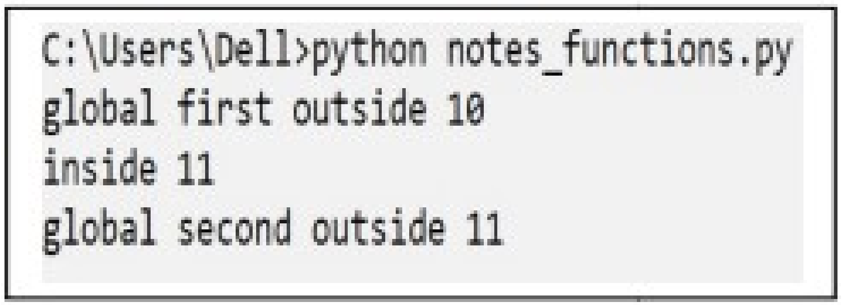
```
C:\Users\Dell>python notes_functions.py
10
Traceback (most recent call last):
  File "C:\Users\Dell\notes_functions.py", line 264, in <module>
    f1()
  File "C:\Users\Dell\notes_functions.py", line 261, in f1
    x=x+1             #UnboundLocalError. Global variable is readonly inside a function. We cannot m
odify the value of that variable inside the function directly
      ^
UnboundLocalError: cannot access local variable 'x' where it is not associated with a value
```

Function: Global and Local variables

Example 3: Usage of global keyword

```
x=10
print("global first outside",x)
def f1():
    global x
    x=x+1
    print("inside",x)
f1()
print("global second outside",x)
```

#If you interchange the first two statements inside the function, it results in Error.



```
C:\Users\Dell>python notes_functions.py
global first outside 10
inside 11
global second outside 11
```

Function Call: Arguments and Parameters

Example 4:- Calculate the sum of two numbers given and return the sum.

```
def add_numbers(x,y):
    sum = x + y
    return sum

output = add_Numbers(10,5)
print("The sum of two numbers is = ",output)
```

Output:

```
The sum of two numbers is = 15
```

Function Call: Arguments and Parameters

Example 5:- Calculate the area of triangle given the dimensions.

```
def area_tri(b,h) :
    a=0.5*b*h
    return a
Area=area_tri(5,6)
print("area=",Area)
```

- The function definition has two parameters, which indicate that it requires two arguments to be passed when the function is called/invoked.
- The arguments are constants in this example, 5 and 6.
- These values are copied to the parameters b and h respectively.

Function Call: Arguments and Parameters

When a function is called, it is imperative that the number of arguments passed **MUST ALWAYS MATCH** the number of parameters in the function definition

Example 6:-

```
def  product(x,y,z) :
          a=x*y*z return a
r=product(5,6,2)
#r=product(5) #error : too few arguments
#r=product(5,6,2,7) #error : too many arguments
print("product =",r)
```

Output:

```
product = 60
```

Function Call: Categories of Functions

1. No arguments : No return value

```
def add():
        a = 10
        b = 20
        print(a+b)
add()

Output: 30
```

1. No arguments : with return value

```
def add()
   a=10
   b=20
   return a+b
sum = add()
print(sum)

Output: 30
```

Function Call: Categories of Functions

3. With arguments : No return value

```
def add(a,b):
        print(a+b)
add(10,20)
```

Output:

```
30
```

4. With arguments : With return value

```
def add(a,b):
    return a+b
sum = add(10,20)
print(sum)
```

Output:

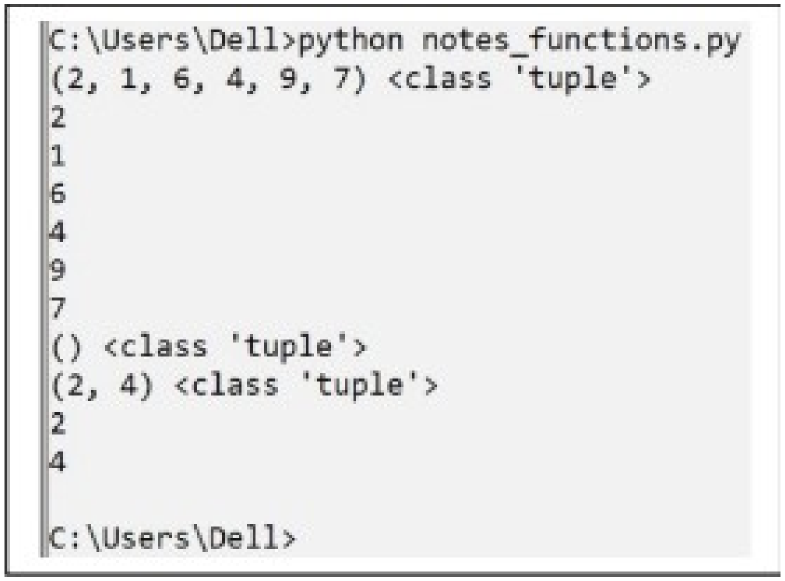
```
30
```

Variable number of positional arguments: *arg [Any variable can be used with *]

- * handles the variable number of positional arguments in the function.
- arg is of type tuple inside the function.

Example 7:

```
def f1(*arg):
    print(arg, type(arg)) #arg is of type tuple
    for i in arg:
        print(i)
f1(2,1,6,4,9,7) #All three function calls are valid
f1()
f1(2,4)
```



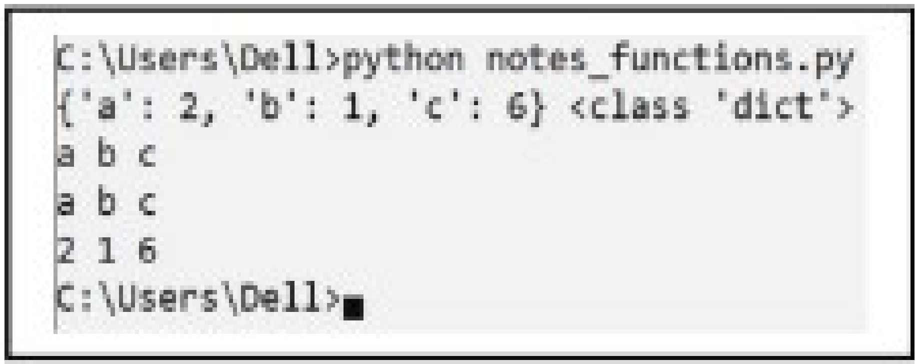
```
C:\Users\Dell>python notes_functions.py
(2, 1, 6, 4, 9, 7) <class 'tuple'>
2
1
6
4
9
7
() <class 'tuple'>
(2, 4) <class 'tuple'>
2
4

C:\Users\Dell>
```

Variable number of keyword arguments: **kwarg [Any variable can be used with **]

- ** handles the variable number of keyword arguments in the function.
- kwarg is of type dictionary inside the function.

Example 8:

```
def f1(**kwarg):
     print(kwarg, type(kwarg)) #kwarg is of type dict
     for i in kwarg:
          print(i,end="")
    print()
    for i in kwarg.keys():
          print(i,end="")
    print()
    for i in kwarg.values():
         print(i,end="")
f1(a=2,b=1,c=6)
```



```
C:\Users\Dell>python notes_functions.py
{'a': 2, 'b': 1, 'c': 6} <class 'dict'>
a b c
a b c
2 1 6
C:\Users\Dell>
```

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai – sindhurpai@pes.edu
Dr. Chetana Srinivas