

## Functions and Parameters

```
def clientFunc():  
  
    # some code  
    res = serverFunc(param1,param2)  
    # some more code  
  
def serverFunc(p1,p2):  
    local = 33+p1 # local variables are known only to the function  
    # code  
    return result
```

### *Important Definitions*

**Function** is a worker-bee that takes in some data (called parameters or arguments), performs some computations, then returns a result. It can also modify parameters (discussed below)

**Scalar value** – an atomic value. Integers, floating point numbers, strings, and booleans.

**Object**-- complex multi-item data, such as a list

**Local Variable** -- a variable whose scope is restricted to a particular function. Temporary, scratch space. Private to a function.

**Parameter** -- Data sent to one function from another.

**Formal Parameter** -- The parameter defined as part of the function definition, e.g., x in:

```
def cube(x):  
    return x*x*x
```

**Actual Parameter** -- The actual data sent to a function. It's found in the function call, e.g., 7 in

```
result = cube(7)
```

or y in:

```
y=5  
sq = square(y)
```

The actual parameters are connected to the formal parameters by order, not by name.

**What effect can a function have on the world?**

It can produce a return value.

It can modify parameters if they are objects (e.g., a list). It cannot modify scalar params.

**Pass-by-Copy parameter passing**-- the value of the actual parameter is copied to the formal parameter. In Python, scalar values are sent by-value. Lists and other objects are sent by reference.

```
def processNumber(x):
    # some code
    # return
    x
    54
    67

# main
y=54
processNumber(y)
y
54
```

**Pass-by-Reference parameter passing**-- a reference to the actual parameter is sent to the function. When we trace a program, and a list is sent, we don't copy the list to the actual parameter box, we draw an arrow:

```
def processList(list):
    # some code
    #return

# main
aList = [5,2,9]
processList(aList)
```

The diagram shows a variable named list with an arrow pointing to a variable named aList which contains the list object [5,2,9]. This illustrates that the function receives a reference to the original list object rather than a copy.

1. Consider the programs below. Trace them with pencil and paper by drawing boxes for each variable and 'executing' each statement of the program. Also, show what is printed out for the program

a .

```
def increment(x):
    x=x+1

# main program
x = 3
print x
increment(x)
print x
```

b. 

```
def increment(x):
    z = 45
    x = x+1
    return x
```

```
# main
y = 3
print y
y = increment(y)
print y
q=77
print q
increment(q)
print q
print x
print z
```

c. 

```
def incrementList(list):
    i=0
    while i<len(list):
        list[i]=list[i]+1
        i=i+1
```

```
# main
list = [1,2,3]
print list
incrementList(list)
print list
```

2. Suppose you were asked to write a function 'switch' which switched the value of two variables, e.g., given:

```
x= 5
y= 7
switch(x,y)
```

The goal is that switch would make  $y=5$  and  $x=7$ . Can you write such a function?

3. Write a function, with pencil and paper, that doubles the value of each element of a list. Write the function twice, once with the list as a parameter, and once with it as a return value.

4. Suppose you were asked to write a function that doubles the value of an integer. Could you do it with the integer as a parameter or as a return value?

5. Trace the following program using boxes and arrows. Trace it directly to the right of the code.

```
def reverse (list, num):  
    size = len(list)  
    i=0  
    while i<num:  
        temp = list[size-1-i]  
        list[size-1-i]=list[i]  
        list[i]=temp  
        i=i+1  
    num=num+1
```

```
mylist=[1,2,3,4,5,6]  
num=3  
reverse(mylist,num)  
print mylist  
print num
```