Parameter passing

#Passing parameters to functions: Read the 3 simple examples below

... In Python, object references are passed by value.

What you pass to a function is an object. When you pass on object to a function, you are actually passing the value of the object.

1. If the object is immutable (say, you pass the value of a variable, or you pass a string) then no matter what you do in the function, you cannot change the value of the object being passed in the caller of the function.

Now type caller1() to see what happens with the two functions below.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

... 2. If the object is mutable (say, you pass a list, but the value of this list object is really all the list elements) then you can change the contents of the list inside the function, and the change is also made in the caller of the function.

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```

Now type caller2() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller1():
    a = 1
    try_to_change(a)
    print(a)  #this should print 1, so a has not changed
```

Now type caller1() to see what happens.''

```python
def try_to_change(x):
    x = x + 1  #this increments the parameter value by 1
    print("--->", x)  #this should print 2, but x is local to this function
    return

def caller2():
    a = [1, 2, 3]
    try_to_change(a)
    print(a)  #this should print [2, 3, 4], so a has been changed
```
def list_append(somelist):
    somelist.append("$")  # append a $ to the end of the input
list
    print("-->",somelist)  # should print incoming list with $ appended
$ appended
    return

def caller2():
    a = ["Our","currency","is","the"]
    print(a)
    list_append(a)
    print(a)  # the function should have appended $ to end of list a
appended $ to end of list a
    object was changed

#
""

3. But you have to understand what goes on with the parameter itself, which
is a local variable
in the function. Reassignment of the value of the object does not change the
object in the caller
but only in the function.

Now type caller3() to see what happens
""

def reassign_list(somelist):
    somelist = ["No","it","is","not","the","$"]

    print("-->",somelist)  # of course the local variable somelist gets the
above assignment
    # but it's only a local assignment. The
value of the list object passed
    # by the caller will not have changed
    return

def caller3():
    a = ["Our","currency","is","the", "$"]
    print(a)
    reassign_list(a)  # the value of the list (its contents) is passed to
the function
    print(a)  # the functions local variable somelist was
reasigned, but variable a
    # in the caller remains untouched