Functions: Redux

A brief discussion of some of the more advances topics/issues associated with functions.

Lists Are References: Copying Lists

- Recap (list variables are actually references to a list):
  - Assigning using just the name of the list just copies the references not the data e.g., list1 = list2
  - To copy the elements of one list to another a loop is needed to copy each successive elements.
- Name of the online example: copy.py

```
list1 = [1,2,3,4]
list2 = []

for i in range (0, 4, 1):
  list2.append(list1[i])

print list1, list2
list1[1] = 99
print list1, list2
```

Passing Simple Types As Parameters: *Pass By Value*

- Passing integers, floats, Booleans as parameters results in a local copy of the parameter being made in the function.
- This parameter passing mechanism is referred to as *pass by value*:
  - Mnemonic aid: A copy of the *value/data* stored in the parameter passed in is what's stored in a local variable of the function that was called.
- The local copy will have the same data as the parameter.
- However the local copy is separate from the parameter so it may change independently from the parameter.
- Alternatively: Changes made to the parameter must be returned back to the caller in order for the changes to be accessible outside of the function.

Passing Simple Types As Parameters: *Pass By Value (2)*

- Name of the online example: parameter1.py

```
def fun1(x):
   x = x + 1

def fun2(x):
   x = x + 1
   return (x)

def main():
   x = 1
   print(x)
   fun1(x)
   print(x)
   x = fun2(x)
   print(x)

main ()
```

Passing Lists As Parameters: Pass By Reference

- Unlike what you've seen with parameter passing so far, modifying a list that's been passed as a parameter to a function *may* modify the original list.
    - It all depends upon how the list is accessed in the function.
- When the reference is passed as a parameter to a function a local reference also refers to the list.
    - The local reference can be reassigned to another list e.g., list1 = list2
    - OR
    - The local reference can be used to change the original list e.g., list1[1] = list2[12]
- This parameter passing mechanism is referred to as *pass by reference*:
    - Mnemonic aid: When a parameter is passed by reference there is a *local variable that refers to the original parameter*.

Original List Is Changed

- Passing lists into functions is done using a different mechanism
    - When a list is passed into the function a local reference variable refers to the original list.
- Name of the online example: parameter2.py

```
def fun (list):
  list[0] = 99
  print (list)


def main ():
  list = [1,2,3]
  print (list)
  fun (list)
  print (list)

main ()
```

Original List Is Unchanged

- If the local reference is assigned to another list then it will obviously no longer refer to the original list.
- (Effect: changes made via the local reference will change the local list and not the original that was passed into the function).
- Name of the online example: parameter3.py

```
def fun (list):
  list = [3,2,1]
  print(list)

def main ():
  list = [1,2,3]
  print(list)
  fun(list)
  print(list)

main ()
```

Parameter Passing: One Last Comprehensive Example

- Name of the online example: parameter4.py

```
def fun1(list1,list2):
   list1 = list2
   print("During fun1:",list1,list2)

def fun2(aList):
   aList = ["Eric","Cart"]
   print("During fun2:", aList)
```

Parameter Passing: One Last Comprehensive Example (2)

```
def fun3(list1,list2):
    list1[0] = list2[0]
    list1[1] = list2[1]
    list1[2] = list2[2]
    print("During fun3:", list1,list2)
    return (list1)

def fun4(list1,list2):
    list1[0] = list2[0]
    list1[1] = list2[1]
    list1[2] = list2[2]
    print("During fun4:", list1,list2)
```

Parameter Passing: One Last Comprehensive Example (3)

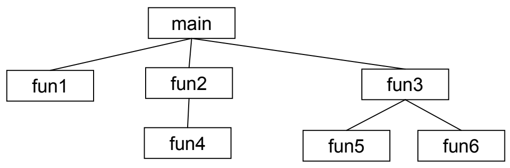
```
def main():
    print("Changes made in function don't persist (example with parameters)")
    list1 = [1,2,3]
    list2 = [3,2,1]
    print("Before fun1:", list1,list2)
    fun1(list1,list2)
    print("After fun1:", list1,list2)
    print()

    print("Changes made in function don't persist (example with local variable)")
    list1 = [1,2,3]
    print("Before fun2:", list1)
    fun2(list1)
    print("After fun2:", list1)
    print()
```

Parameter Passing: One Last Comprehensive Example (4)

```
    print("Changes made to original list using return value")
    list1 = [1,2,3]
    list2 = [3,2,1]
    print("Before fun3: ", list1,list2)
    list1 = fun3(list1,list2)
    print("After fun3:", list1,list2)
    print()

    print("Changes made to original list using reference parameters")
    list1 = [1,2,3]
    list2 = [3,2,1]
    print("Before fun4:", list1,list2)
    fun4(list1,list2)
    print("After fun4:", list1,list2)

main()
```

Where To Declare Your Variables

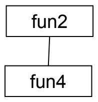
- In a program with many functions it must be determined in which function should a variable be created.
  - Main calls fun1, fun2, fun3



- Rule of thumb:
  - To minimize the potential for side-effects: Do not declare a variable any higher in the hierarchy than needed (as low as possible).
- Simple case:
  - If a function is only needed in a bottom level function (fun1,2,4,6) then it should be declared as local to that function.

Where To Declare Your Variables (2)

- Other cases:
  - If a variable must be passed as a parameter into a function then the variable must be declared in the caller of that function.
  - Example: fun2 calls fun4



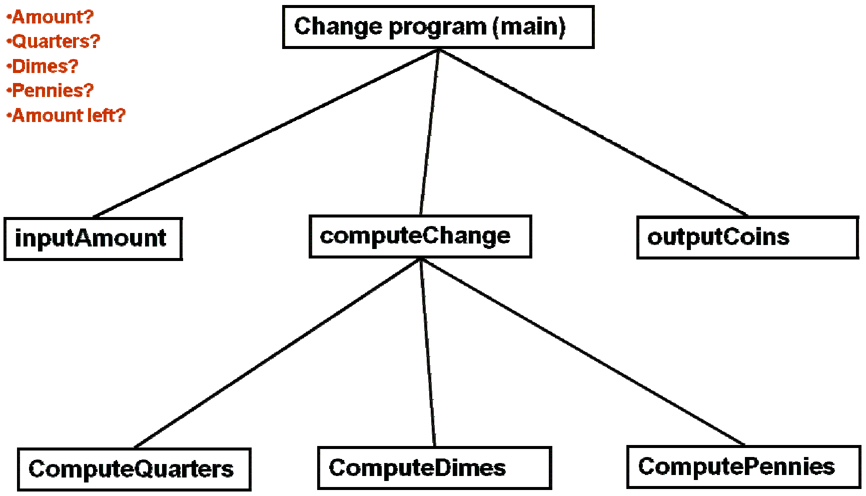
  - If a variable in fun2 must be passed as a parameter to fun4, then that variable must be created in fun2 (higher if that parameter is passed into fun2).

Error!
```
def fun2():
   fun4(num)

def fun4(num):
   print(num)
```

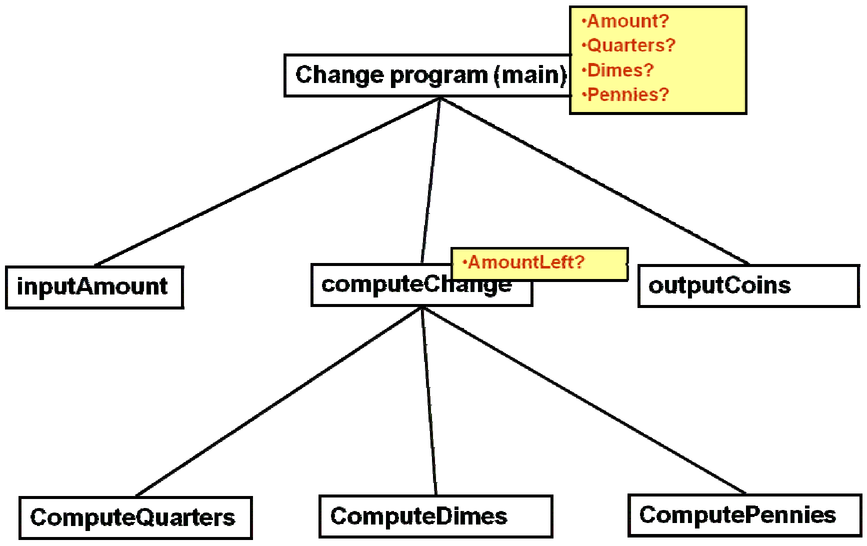
OK:
```
def fun2():
   num = 12
   fun4(num)

def fun4(num):
   print(num)
```

Example: Where To Declare Your Variables

- Amount?
- Quarters?
- Dimes?
- Pennies?
- Amount left?



Example: Where To Declare Your Variables (2)



After This Section You Should Now Know

- The difference between pass by reference and pass by value
- When a reference parameter does and does not change the original data
- Some guidelines for where you should declare your variables in a hierarchy of functions that