Functions (2)

CS 1111
Introduction to Programming
Spring 2018
Print versus Return

Print

• All **print** statements reached by the function are executed
  • They are printed to the screen
• After a print statement is executed, the execution proceeds to the next statement

Return

• A **return** statement is optional
• Only first return statement reached gets run
• If no return statement, function returns **None**
• A **return** ceases execution of a function and returns a value
  • At most one (the first) **return** statement that is reached during a particular function call is executed
• A return value is not printed, unless a function is printed
  • **print**(add(2, 3))
## Void versus Value-Returning Functions

<table>
<thead>
<tr>
<th>Void functions</th>
<th>Value-Return functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Does not return anything</td>
<td>• return something</td>
</tr>
<tr>
<td>• <em>None</em> in Python</td>
<td></td>
</tr>
<tr>
<td>• Examples</td>
<td>• Examples</td>
</tr>
<tr>
<td>• print(str)</td>
<td>• abs(some_number)</td>
</tr>
<tr>
<td>• random.seed(seed)</td>
<td>• random.randInt(0, 100)</td>
</tr>
<tr>
<td>• random.shuffle()</td>
<td>• random.sample(some_list)</td>
</tr>
</tbody>
</table>

(We will talk about random module later)
## Rule 1
Variables and items on the heap are stored in separate locations.

## Rule 2
A primitive type is stored directly with its variable; a complex type (such as a list) has its variable store a memory address, and that memory address refers to a location on the heap where the actual data lives.

## Rule 3
Every assignment begins by either creating a variable space (and heap location, if necessary), or emptying out the existing contents of a variable space (**but NOT the heap**!), and then copying either a value or memory address from one box into the other. A variable or memory location must only store either numbers/booleans, or a memory address, NEVER the name of a variable.

## Rule 4
1. Make space for the function.
2. Look at the function definition and make space for its arguments (if any).
3. Copy the values from the function call into the space created in (2). Remember these are assignments (see the rules from the previous chapter for how to handle assignments).
4. Complete the body of the function. Remember to only refer to variables local to the function you created in (1).
5. Circle the return value; if no return value, circle `None` (in Python).
6. Cross out all local variables (except the return) to remind you they will disappear; however, **NOT** touch the heap!
7. Cross out the function call and replace it with the value circled in (5).

## Rule 5
**Only a print statement generates output (a return statement does not).**

## Rule 6
Continued from Rule 3, the left hand side of an assignment must simplify to a location in memory in order to make the assignment. The right hand side must simplify to either a constant (like a number or True/False) or memory address (for complex types like lists).

---

Based in part on “Agnostic Programming: Learning to Design and Test Basic Programming Algorithms” by Kinga Dobolyi, Kindle]
Pass by Value

my_value = 11

def change_a_value(some_value):
    print("Inside change_a_value(), some_value starts as: ", some_value)
    some_value *= 2
    print("some_value now is: ", some_value)

print("Starting the program, my_value starts as: ", my_value)
change_a_value(my_value)
print("my_value now is still: ", my_value)

• Passing immutable types to a function.
• A copy of the variable (value and everything) is sent to the function.
• Changes made to the variable passed in are not reflected back where the function was called.
Pass by Reference

my_list = ['a', 'b', 'c', 'd']

def change_a_ref(some_list):
    print("Inside change_a_ref(), some_list starts as: ", some_list)
    some_list.append('x')
    print("some_list now is:", some_list)

print("Starting the program, my_list starts as: ", my_list)
change_a_ref(my_list)
print("my_list now is:", my_list)

• Passing **mutable types** to a function.
• A copy of the **memory address** of the object, is sent to the function.
• Changes made to the variable passed in are reflected back where the function was called.
```python
def main():
    print('The sum of 12 and 45 is ')
    show_sum(12, 45)

def show_sum(num1, num2):
    result = num1 + num2
    print(result)

main()
```

The sum of 12 and 45 is
57
Local and Global Variables

- **Local variables**
  - The arguments, and any variables declared in the function
  - Cannot be seen by other functions or code
  - Arguments given values in function call
  - Even if they have the same name as variables outside the function, the computer treats them as different (think of two people both named Tom; they are different people though they happen to be named the same)

- Each function call has its own memory space and variables
- These local data disappear when the function finishes
- Arguments are assigned from the function call

- **Global variables**
  - Is accessible to all the functions in a program file
Local Variables

number = 0

def main():
    number = int(input('Enter a number: '))
    show_number()

def show_number():
    print('The number you entered is ', number)

main()

Enter a number: 7
The number you entered is 0
Global Variables

number = 0

def main():
    global number
    number = int(input('Enter a number: '))
    show_number()

def show_number():
    print('The number you entered is ', number)

main()

Enter a number: 7
The number you entered is 7
def my_function(name="Mary", school="UVa"):
    print(name + " goes to " + school)

# call a function without param passing
my_function()

# call a function with some params (ordering)
my_function("Tom")

# call a function with a named param
my_function(school="GMU")

# call a function with param passing (match number of params and ordering)
my_function("Ann", "VT")

# call a function with named params in any order
my_function(school="GMU", name="Ann")
Importing Existing Functions

- Assuming the add() function and other functions are saved in a file called math_lib.py (i.e., module’s name is math_lib)

- Import module_name or From module_name import * allows us to import all functions from math_lib.py into the current file
  - Call functions from other files
  - Can use add() without defining it here

- Python imports some standard functions, such as str() and len() automatically; others need the import statement

```python
import math_lib
math_lib.add(2, 3)
print(math_lib.add(4, 5))
print(math_lib.add(1, -1))
```

```python
from math_lib import *
add(2, 3)
print(add(4, 5))
print(add(1, -1))
```