Functions Part II

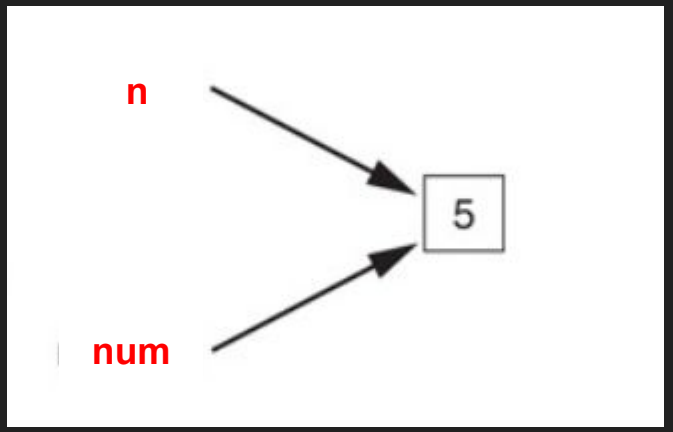
Passing Arguments to functions (review)

```
def some_function (num):
    if num == 5:
        print ("This is a five.")
    else:
        print ("This is not a five.")


def main ():
    n = int(input("Please enter a number: "))
    some_function(n)

main()
```

The variable n and the parameter num reference the same value.

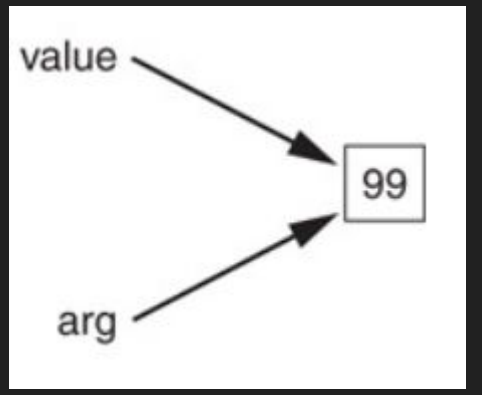


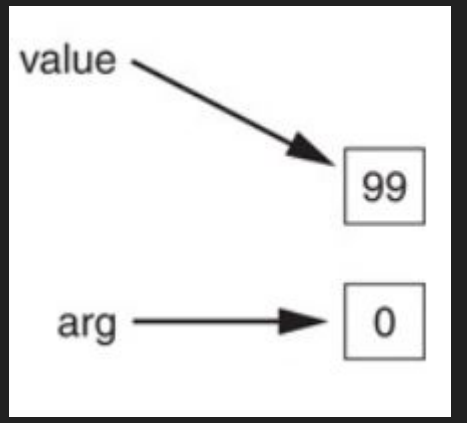
Making Changes to Parameters

```
def main():
    value = 99
    print('The value is: ', value)
    change_me(value)
    print('Back in main the value is: ', value)

def change_me(arg):
    print('I am changing the value.')
    arg = 0
    print('Now the value is', arg)

# Call the main  function.
main()
```





Even though the parameter variable arg was changed in the change_me function, the value variable in the main was not modified.

Python Tutor

http://pythontutor.com/visualize.html#mode=display

(Try running change_me.py and other code here)

Local and Global Variables Review

- The ***scope*** of a variable is everywhere it is defined
- Variables created *inside* a function have meaning only inside that function - ***local*** variables
  - The function is their ***scope***
- Variables defined outside any function are called ***global*** variables
  - Their scope includes all functions that declare the global variable

Example of local variable scope

```
def get_name():
    name = input('Enter your name: ') #name is a local variable

def main():
    get_name()
    print('Hello', name)        #Name is out of scope!
                                #Will get error saying name is not defined

main()
```

Example of global variable scope

```
# Create a global variable.
number = 0

def main():
    global number #using global keyword to declare number variable
                  #because going to assign a value to global variable inside main
    number = int(input('Enter a number: '))
    show_number()

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

main()
```

What would happen if you commented out the line:
global number
?

Value-Returning Functions

Last lesson you learned about void functions. A void function is a group of statements within a program that perform specific task.

A value-returning function returns a value back to the part of the program that calls it.

The value that is returned can be assigned to a variable, displayed on the screen, or used in a mathematical expression.

Two Kinds of Functions

- ***Void*** functions
  - Executes its code and terminates
  - Does not return a value
  - E.g.: print()
- ***Value-returning*** functions
  - Executes its code and returns a value to the statement that called it.
  - E.g., **number** = random.randint(n)
  - E.g., **userInput** = input(“Type your input: ”)
  - Ends with ***return*** statement.

Value-Returning Functions

def *function_name*():

    statement

    statement

    etc.

    return ***expression***

the value of ***expression*** that follows the keyword return will be sent back to that part of the program that called the function.

Function *ends* after return keyword. Try printing something after return and see what happens!

Functions Can Return Values

- Functions can return any type of value
  - integer number
  - floating point number
  - string
  - boolean (True or False)

  - multiple values (not covering this right now)

Comparing two types of add function

value-returning add function:

```
def sum (num1, num2):
    total = num1 + num2
    return total

def main():
    result = sum(2,3)
    print('Total is: ', result)

main()
```

void add function:

```
def sum(num1, num2):
        total = num1 + num2
        print('Total is: ',total)

def main():
    sum(2,3)

main()
```

Remember -> value returning functions involve setting a variable!

If you don’t set it to a variable, you_will_lose_it!

#1 mistake made by students with value-returning functions.

So set/assign it to a variable E.g.:

```
total = sum(2,3)
userInput = input(“Type your input: ”)
number = random.randint(n)
```

Returning the value of an expression

```
def sum (num1, num2):
    total = num1 + num2
    return total
```

```
def sum (num1, num2):
    return num1 + num2
```

Value-returning functions can simplify code and reduce duplication, e.g.:

```
def discount(price):
    return price * DISCOUNT_PERCENTAGE


def main():
    sale_price = reg_price - discount(reg_price)
```

Boolean Functions

Returns **True** or **False**

(Can also be a *task* that returns **True** if successful and **False** if unsuccessful)

Re-write your isEven(n) function so that it returns True or False depending on whether n is even or odd.

Call isEven(n) in your main function and print the result.

isEven(n)

```
def isEven(n):
   if n%2==0:
      return True
   else:
      return False

def main():
   num = int(input('Enter the number: '))
   result=isEven(num)
   print(result)

main()
```

Pre-Defined Functions

- Standard functions (built-into python)
  - E.g., print(), input(), range(), int()
- Other pre-defined functions (modules)
  - Must *import* their module before you can use them, e.g.:

```
import math  #library of math functions & global variables
import random   #library of randomize functions
r = 14
area = math.pi * r**2          #uses math's pi global variable
x = math.factorial(20)                    #calls factorial function
number = random.randint(1, 20)    #calls randint function
```

Function *Signature*

- def <function name>(p1, p2, p3, …):
- A **function signature** defines input and output of functions. A **signature** can include: parameters and their types or a return value and type.
- Can have multiple functions with same name, if they have different signatures
  - def printName(first, middle, last):
  - def printName(first, last):

Organizing Functions in Modules

Put related function definitions in one .py file, with no main program

To use those functions in a program:

- Put module (.py) file in directory with main program
- At top of main program, import the module(s) you need.
- Call the imported functions in your main program, using dot notation, e.g. <modulename>.<function>()
  - Different modules can have functions with same name, because dot notation distinguishes them

Modules can define global variables, e.g., pi