# Handout 5

## Defining functions.

A **function** is a collection of statements that are grouped together to perform an operation.
Advantages from defining and using functions:

- improve the quality of the program by modularizing the code
- reduce redundant coding and enable code reuse
- enable division of labor

A function definition consists of a **header** and a **body**.

The header begins with the **def** keyword, followed by function's name and parameters, followed by a colon.

The body consists of the indented code underneath the header.

The function definition does not execute the function body; function gets executed only when the function is called.

```
# The following is a function definition
# This function has no parameters and no return value
def printHelloWorld():
    print ('Hello, world')

# Call the above function three times
printHelloWorld()
printHelloWorld()
printHelloWorld()
```

The variables defined in the function header are known as *formal parameters*.
When a function is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*:

```
#-------------------------------------------
# Function takes a single parameter; returns no value
def printHello (name):
    print ('Hello,' , name , '!')

printHello('class')
aname = input('Greetings! what is your name?')
printHello(aname)
```

A function may return a value using the **return** keyword.
A function that does not return a value returns a special value, **None** (as the two functions above)

```
#-------------------------------------------------------
# Function takes a single parameter, returns a single value

def helloString (name):
    result = 'Hello,' + name + '!'
    return result

greetingString =  helloString('Joe')
```

**Practice problem:**

- identify function calls vs function definition
- identify function parameters, return value, local and global variables
- run this code line-by-line in **Visual Debugger**

```
def calcTotal(numberItems, unitCost):
    totalCost = numberItems * unitCost
    return totalCost


count = int(input('Enter # of books: '))
total = calcTotal(count, 17.50)

print('Total: $', total)
```

## PRACTICE PROBLEMS ON FUNCTION DEFINITIONS

1. Define a function `percentageGrowth`, which has 2 input parameters that are floating point values representing old and new value of some metric (e.g. stock price, average grade, etc..) . The function should calculate and return the percentage of growth from old value to the new one.

   Create a main function in which you test the `percentageGrowth()` function.

2. Define a function `validTime()` which has 1 input parameter that is a string. If the string represents a valid time value based on the 24-hour clock and expressed in HH:MM or H:MM format, return `True`, otherwise, return `False`.

   Create a main function in which you test the `validTime()` function by asking the user to enter a time string until a valid value is entered.

## RETURNING MULTIPLE VALUES (AS A TUPLE)

- Python offers a simple way to return multiple results from a function
  - The return statement includes comma-separated items
  - In reality, there is one value returned, but it is a *tuple* that combines all values

```
def calcTotal(numberItems, unitCost, taxRate):
    pretaxCost = numberItems * unitCost
    aftertaxCost = round(pretaxCost * (1 + taxRate), 2)
```

```
    return pretaxCost, aftertaxCost

# can assign the return value to a single variable - tuple with 2 values
result = calcTotal(4, 1.99, .05 )
print (result)

# can assign the return value to as many vars as values returned
notax, taxed = calcTotal(4, 1.99, .05 )
print(notax)
print(taxed)
```

3. Define a function `extractTime()` which has 1 input parameter that is a string. Check that the string represents a valid time value using function `validTime()` developed earlier. If so, return two integers: the hour and the minute value specified in the input string. If the string is not a valid time specification, return 0, 0.

   Create a main function in which you test the `extractTime ()`.

## DEFINITIONS WITH DEFAULT PARAMETER VALUES

When there is a good default for a parameter, you can specify that default.
Then you can call the function with the default or other value (remember print's **sep** argument?)

```
# define the function, including an argument with the default value
def calcTotal(numberItems, unitCost, taxRate = .05):
    pretaxCost = numberItems * unitCost
    aftertaxCost = round(pretaxCost * (1 + taxRate), 2)
    return aftertaxCost

# inputs
count = int(input('Enter # of books: '))
cost = float(input('Enter cost per book ($): '))

#can call with or without parameter taxRate
print (calcTotal(count, cost))
print (calcTotal(count, cost, taxRate = .075)) #override the default
print (calcTotal(count, cost, .075 ))  #same as line above
```

## SCOPE OF VARIABLES

Scope: the part of the program where the variable can be referenced.

A variable created inside a function is referred to as a *local variable*. Local variables can only be accessed inside a function. The scope of a local variable starts from its creation and continues to the end of the function that contains the variable.

In Python, you can also use *global variables*. They are created **outside all functions** and are accessible **to all functions** in their scope.

**The use of global variables should be minimized**, to include only global **constants**, which should be declared in all capital letters.

Global variables make it difficult to trace code and debug the program, since they can be changed by any part of the code. Local variables can only be changed within the methods, in which they participate.

Here's the design that places all code in functions: notice there is a single function call to main(), everything else is defined via and within functions. The name main() has **no special significance** in Python (differently from C, Java).

```
# define the function
def calcTotal(numberItems, unitCost):
    totalCost = unitCost * numberItems
    return totalCost

def main():

    count = int(input('Enter # of books: '))
    total = calcTotal(count, 17.50)
    print('Total: $', total)

main()
```

Review the following program:

```
# prompt user for inputs
def getSalesInfo():
    count = int(input('Enter # of books: '))
    cost = float(input('Enter cost per book ($): '))
    return count, cost

# calculate the pre-tax cost and tax
def calcTotal(numberItems, unitCost, taxRate):
    pretaxCost = round(numberItems * unitCost, 2)
    salesTax = round(pretaxCost * taxRate, 2)
    return pretaxCost, salesTax

def displaySales(pretax, tax):
   print('Pre-tax: $', pretax)
   print('Tax:     $', tax)
   print('Total:   $', pretax+tax)

# call each function for input, processing, and output
def main():
   count, cost = getSalesInfo()
   pretax, tax = calcTotal(count, cost)
   displaySales(pretax, tax)

main()
```

**PRACTICE PROBLEM:**

4. *Hint*: **you will need to know how to use string function split() – see Handout 4**

A good way to generate a password is to create it from a familiar phrase. Define a function `passwordFromText()` – which has one string parameter, let's call it *sentence*. The function must compose and return a password generated from the sentence using the following algorithm. The password must include every first letter of each word (word is a sequence of non-blank characters that starts with a letter) followed by every non-alphanumeric value that is not a space. Furthermore,
- If the password composed this way is longer than 12 characters, cut it to the first 12 characters.
- If the password ends up having no digits and no special symbols, replace one of the characters in it with a '*' and one other randomly chosen character with number 7

For example, given sentence `'Out, out brief candle!'` the function should return `'Oobc,!'`

Passed sentence `'He took his shoes off 20 years ago.'` the function should return `'Hthso2ya.'`

Here's another example (based on a quote attributed to Steve Jobs): `'I'm convinced that about half of what separates the successful entrepreneurs from the non-successful ones is pure perseverance. ',` should yield a string like `'Icta7owst*ef'` -in which 7 and * are replacing original letters in *randomly* chosen positions, so your password can be different.