Workshop 4

Functions

A function is a group of statements that can be invoked from anywhere in a program where the program knows that the function exists. In other words, a function can be invoked in any namespace that knows about the function.

Functions are defined using a **def** statement. The end of the statements in a function is determined by the indention. Functions are invoked by simply using the function name in the code. A function terminates execution when it reaches the end of the statements in the function or a **return** statement is encountered.

A return statement can specify an object to be "returned" by the function to the module that invoked the function. Alternatively, a return statement can specify that nothing is returned by returning the **None** object or by specifying no object at all. A return statement can appear anywhere in a function and a function may have multiple places where a return statement is used.

Generally, the structure of a function looks like:

def *functionname* (optional arguments):
    statements

def *functionname* (optional arguments):
    statements
    return

def *functionname* (optional arguments):
    statements
    return None

def *functionname* (optional arguments):
    statements
    return *object*

def *functionname* (optional arguments):
    statements
    return

- **def** is executable code.
- **def** creates an object and assigns it to a name.
- **return** sends a result object back to the caller.
- **global** declares module-level variables that are to be assigned.
- Arguments ae passed by assignment (object reference).
- Arguments are passed by position, unless you say otherwise.
- Arguments, return values, and variables are not declared.

A simple function:

```
def times (x, y):
        return x * y
```

This function multiples the two arguments passed to it and returns the result.

```
Q = times (2, 4)
```

This invokes the times function, passes the 2 to the x and the 4 to the y, and returns the value of 8 to be assigned to Q.

Function Scope

Scope defines where an object name exists. In Python, a namespace is associated with each package of code. For our purposes, there are namespaces for the main program module, any imported modules, and functions.

By default, all names assigned inside a function are associated with that function's namespace, and no other.

If a variable X is assigned a value in a main program and a variable X is assigned a value in a function that is defined in that main program, the X in the function is a different variable than the X in the main program (unless you explicitly include a **global X** statement in the function).

- If a variable is assigned inside a def, it is *local* to that function.
- If a variable is assigned outside all defs, it is *global* to the entire file.

```
X = 99
def func():
        X = 88    # Different X
```

```
X = 99
def func():
        global X
        X = 88    # Same X
```

- Global names are variables assigned at the top level of the enclosing module file.
- Global names must be declared only if they are assigned within a function.
- Global names may be referenced within a function without begin declared.

The use of global variables is discouraged in normal situations.

Arguments

Arguments are passed by automatically assigning objects to local variable names.
Assigning to argument names inside a function does not affect the caller.
Changing a mutable object argument in a function may impact the caller.

Immutable arguments are effectively passed **"by value"**. You can think of this as meaning a copy of the object is used in the function. Thus, the object that is passed into the function will be the same after the function executes as it was before the function executes.

Mutable arguments are effectively passed **"by reference"**. As in all cases we have seen where a change is made to a mutable object, every other object that references the mutable object will see the change.

```
>>>def f(a):
…       a = 99

>>>b=88
>>>f(b)
>>>b

88      ← b is immutable

>>>def f(a):
…       a = 99

>>>L = [1, 2, 3]
>>>f(L[1])
>>>L
[1, 2, 3]       ← L[1] is 2 which is immutable

>>>def f(a, o):
…       a[o] = 99

>>>L = [1, 2, 3]
>>>f(L,1)
>>>L
[1, 99, 3]      ← L is mutable, so it can be changed

>>>def f(a, o):
…       a = a[:]
…       a[o] = 99

>>>L = [1, 2, 3]
>>>f(L,1)
>>>L
[1, 2, 3]       ← A copy of L was made in f, so changes were made to the copy
```

By default, arguments are matched by position, from left to right, and you must pass exactly as many arguments as there are argument names in the function header.

However, you can also specify matching by name and provide default values.

Method	Explanation
Positional: matched from left to right	The normal case is to match passed argument values to argument names in a function header by position, from left to right.
Keywords: matched by argument name	Callers can specify which argument in the function is to receive a value by using the argument's name in the call, with the name=value syntax
Defaults: specify values for optional arguments that are not passed	Callers can specify default values for arguments to receive if the call passes too few values, with the name=value syntax

```
func(value)          Normal argument: matched by position
func(name=value)     Keyword argument: matched by name

def func(name)       Normal argument: matches any passed value by position or name
def func(name=value) Default argument value, if not passed in the call
```

In a function **header**, the arguments must appear in this order:

- any normal arguments (`name`)
- any default arguments (`name=value`)

```
>>> def f(a,b,c): print(a,b,c)
...
>>> f(1,2,3)
1 2 3

>>> f(c=4,a=5,b=6)
5 6 4

>>> f(7,c=3, b=2)
7 2 3
```

```
>>> def func():
...     X = 88
>>> X = 99
>>> func()
>>> X
99
>>> def func():
...     global X
...     X = 88
...
>>> X = 99
>>> X
99
>>> func()
>>> X
88
>>> def f(a):
...     a = a + 10
...
>>> b = 9
>>> f(b)
>>> b
9
>>> def f(a):
...     print('a before the addition:', a)
...     a = a + 10
...     print('a after the addition:', a)
...
>>> b
9
>>> f(b)
a before the addition: 9
a after the addition: 19
>>> def f(a):
...     a = a + 10
...     return a
...
>>> b
9
>>> f(b)
19
>>> b
9
>>> b = f(b)
>>> b
19
>>>
```

Let's write some code!

Rewrite your program from Workshop 3 to calculate the average score in a function. Define the function to have two arguments (score and number of games) and return the average.

```
#This program reads the data in AU gamedata.csv and calculates averages

import csv

def myavg(score, games):
      return score / games

cvscore, chscore, cvnum, chnum = 0, 0, 0, 0
nvscore, nhscore, nvnum, nhnum = 0, 0, 0, 0
tvscore, thscore, tvnum, thnum = 0, 0, 0, 0

#Read the data from the file to be read
reader = csv.reader(open('AUGamedata.csv', 'r'), delimiter=',')

#Process each row
for row in reader:
      datarow = list(row)
      score = int(datarow[3])
      if datarow[1] == 'C':
            if datarow[2] == 'V':
                  cvscore += score
                  cvnum += 1
            else:
                  chscore += score
                  chnum += 1
      elif datarow[1] == 'N':
            if datarow[2] == 'V':
                  nvscore += score
                  nvnum += 1
            else:
                  nhscore += score
                  nhnum += 1
      else:
            if datarow[2] == 'V':
                  tvscore += score
                  tvnum += 1
            else:
                  thscore += score
                  thnum += 1
```

```
print('Average points per game')
print('Home team, nonconference: %4.2f \tNo of games: %s' % (myavg(nhscore, nhnum), nhnum))
print('Home team, conference: %4.2f \t\tNo of games: %s' % (myavg(chscore, chnum), chnum))
print('Home team, tournament: %4.2f \t\tNo of games: %s'% (myavg(thscore,thnum), thnum))
print('Visiting team, nonconference: %4.2f \tNo of games: %s' % (myavg(nvscore, nvnum), nvnum))
print('Visiting team, conference: %4.2f \tNo of games: %s' % (myavg(cvscore, cvnum), cvnum))
print('Visiting team, tournament: %4.2f \tNo of games: %s' % (myavg(tvscore, tvnum), tvnum))

vscore = cvscore + nvscore + tvscore
vgames = cvnum + nvnum + tvnum
hscore = chscore + nhscore + thscore
hgames = chnum + nhnum + thnum
tscore = vscore + hscore
tgames = vgames + hgames

print('Overall home team average: %4.2f \tNo of games: %s' % (myavg(hscore, hgames), hgames))
print('Overall visiting team average: %4.2f \tNo of games: %s' % (myavg(vscore, vgames), vgames))
print('Overall average: %4.2f \t\t\tNo of games: %s' % (myavg(tscore, tgames), tgames))
```

```
#This program reads the data in AU gamedata.csv and calculates averages using sets and a function.

import csv

def myavg(gamelist, games):
      score = 0
      for gamenumber in games:
            score += int(gamelist[gamenumber][3]) #get score from game in gamelist
      num = len(games)
      return score / num, num

#Read the data from the file to be read
reader = csv.reader(open('AUGamedata.csv', 'r'), delimiter=',')

conset, nonconset, tourneyset = set(), set(), set()
visitset, homeset = set(), set()
gamelist = []

i = 0
for row in reader:

      game = list(row)
      gamelist.append(game)

      if game[1] == 'C': conset.add(i)
      elif game[1] == 'N': nonconset.add(i)
      else: tourneyset.add(i)

      if game[2] == 'V': visitset.add(i)
      else: homeset.add(i)

      i +=1


convset = conset & visitset           #set intersections
conhset = conset & homeset
nonconhset = nonconset & homeset
nonconvset = nonconset & visitset
touvset = tourneyset & visitset
touhset = tourneyset & homeset
```

```
allset = conset | nonconset | tourneyset    #union of all sets

print('Average points per game')

print('Home team, nonconference: %4.2f \tNo of games: %s' % (myavg(gamelist, nonconhset)))
print('Home team, conference: %4.2f \t\tNo of games: %s' % (myavg(gamelist, conhset)))
print('Home team, tournament: %4.2f \t\tNo of games: %s'% (myavg(gamelist,touhset)))

print('Visiting team, nonconference: %4.2f \tNo of games: %s' % (myavg(gamelist, nonconvset)))
print('Visiting team, conference: %4.2f \tNo of games: %s' % (myavg(gamelist, convset)))
print('Visiting team, tournament: %4.2f \tNo of games: %s' % (myavg(gamelist, touvset)))

print('Overall home team average: %4.2f \tNo of games: %s' % (myavg(gamelist, homeset)))
print('Overall visiting team average: %4.2f \tNo of games: %s' % (myavg(gamelist, visitset)))
print('Overall average: %4.2f \t\t\tNo of games: %s' % (myavg(gamelist, allset)))
```

One big example (Two pages of code. Functions on this page; main program on next. This is a demo only. Don't code it.):

```
#This program reads input from a user and creates a list of
#variables to be used in the Chernoff face program

import csv

def listlength (num, variablelist):
    if num < 18:
        needed = 18 - num
        print(str(num), 'variables input. ', str(needed), 'more needed.')
    elif num > 18:
        print(str(num), 'variables input. First 18 will be used.')
        variablelist = variablelist[:18]
    else:
        print('18 variables input.')
    print('\n')
    return variablelist

def timestamp ():
    from datetime import datetime, date, time
    t = datetime.now()
    s = t.strftime('%Y') + t.strftime('%m') + t.strftime('%d')
    s = s + t.strftime('%H') + t.strftime('%M') + t.strftime('%S')
    return s

def instruct ():
    print('Enter a variable, a list of variables, or a file path to a .csv file of variables.')
    print('Enter "list" to list variables in the current list.')
    print('----- "purge" to remove all variable from the current list.')
    print('----- "remove" followed by a variable to remove the variable from the list or')
    print('      "remove" followed by a list of variables separated by commas to remove ')
    print('      "multiple variables from the current list.')
    print('----- "help" to get these instructions.')
    print('----- "exit" to exit variable input.')
    return None
```

```
varlist = []
while True:
     reply = input("Enter: ")
     if reply == 'exit': break
     if '.csv' in reply:
          try:
          #Read the data from the file to be read
               reader = csv.reader(open(reply, 'r'), delimiter=',')
               #Process each row
               for row in reader:
                    for var in row: varlist.append(var)                         #builds a list of variables
          except:
               print('File not found:', reply)
     elif reply == 'list':
          print(varlist)
     elif reply == 'purge':
          varlist = []
     elif reply == 'help':
          instruct ()
     elif 'remove' in reply:
          reply = reply.replace('remove','')   #delete the keyword
          reply = reply.replace(' ','')        #delete any spaces
          templist = list(reply.split(','))    #create a list
          for var in templist:
               try:
                    varlist.pop(varlist.index(var))
               except:
                    print(var, 'not found in variable list.')
     else:
          if ',' in reply:
               reply = reply.replace(' ', '')  #remove blanks
               templist = list(reply.split(','))
               for var in templist: varlist.append(var)
          else:
               varlist.append(reply)

     varlist = listlength(len(varlist), varlist)    #check length of list

#output final variable list to a file for future reference
path = 'Varlist' + timestamp() + '.csv'
fout = open(path, 'w')
s = ''
for v in varlist: s = s + v + ','
fout.write(s[:-1])
```