Lesson 1 Introduction to functions

# Learning goals

1. Describe the differences between procedures and functions
   1. Functions calculate and return a value; procedures make a change to the state of a program
   2. Function calls occur on the right side of an assignment statement; procedure calls occur on their own.
   3. Function definitions have a return statement; procedures don’t.
2. Define and use these terms: function definition, parameter, argument, return statement, function call
3. Give examples of functions built into Python: len, str, input, create\_oval, sqrt, int
4. Give examples of procedures built into Python: print, .update, .append, .remove, sleep
5. Write simple functions with parameters and test them using function calls on actual arguments,   
   e.g. getDistance(x1,x2,y1,y2), isEven(x), etc.

# Agenda

1. Functions and procedures you already know in Python
2. The differences between functions and procedures
   1. calculate and return a value vs. change the state of the program
   2. called as the right side of an assignment statement vs. being called in isolation
3. Defining a new function: getAverage( a, b )
4. Anatomy of a function definition:
   1. name
   2. parameters
   3. body
   4. return-statement
5. How a function call works
6. Example 2: Defining a function that returns a boolean instead of a number
7. Python broadcast demo: getAverage, isEven
8. Students practice on *Function Exercises.doc*

## Python commands you know

|  |  |
| --- | --- |
| **print**(“hi”)  screen**.update()**  **sleep(**0.5**)**  myMarks**.append(** 90 **)**  myMarks**.remove(**45**)** | x = **int(**6.5**)**  y = **str(**6.5**)**  z = **range(**0,5**)**  a = **randint(**0,100**)**  b = **sqrt(**64**)**  c = **input(** “Enter your name”)  d = **len(** myMarks **)**  box =screen**.create\_oval(** 0, 0,10,10,fill=”red”) |
| These are **procedures** | These are **functions** |
| Procedures don’t return any calculated values | Functions return a calculated value |
| Procedures change the state of the program. For example, by:   * changing the screen * making the program pause or resume * changing the values of variables  (like myMarks) | Functions should not change the state of the program   * They should not print or draw to the screen |
| Procedures are never part of an assignment statement. There is no equals sign and no LHS. | Function calls are always on the RHS of an assignment statement, and assigned to a variable on the LHS |

## Making our own getAverage function

Wouldn’t it be cool if we could do something like this:

x = randint(1,100)

y = randint(1,100)

avg = **getAverage**( x, y ) #There is no Python command called getAverage(). So let’s make one!

### New program

**def** **getAverage**( num1, num2 ): #num1 and num2 are the function’s ***parameters***

sum = num1 + num2 #These three lines are called the ***body*** of the function

avg = sum/2

**return** avg

z = **getAverage**( 100, 300 ) #An example ***function call***. Its ***arguments*** are 100 and 300

print( z ) #prints 200 to the screen

When Python sees the line z = **getAverage**( 100, 300 ), it does the following:

1. It asks, “Is there a command in my database called getAverage?”
2. If yes, Python assigns the arguments 100 and 300 to the parameters num1 and num2.
3. Then it executes all the lines in the body of getAverage, using 100 and 300 in place of the parameters num1 and num2.
4. It returns the value of avg (200) to the function call. That means Python substitutes 200 in place of getAverage(100,300), so that z now has the value 200.
5. Python treats this exactly as if we had typed z = 200 directly in our original program.

## Example two: a function that returns a boolean

**def** **isEven**( num ): #num is the function’s parameter

if num % 2 == 0:

y = **True**

else:

y = **False**

**return** y

z = **isEven**( 21 ) #An example **function call**. Its argument is 21

print( z ) #Prints “False” to the screen.

#Function call that doesn’t assign to a variable

**for** row **in** **range** (1,9):

y = 150\*row

**for** col **in** **range**(1, 9):

x = 150\*col

**if** **isEven**( row + col ): #Function call to **isEven**. The argument is row+col.

checkerSquareColour = **“red”**

**else**:

checkerSquareColour = **“black”**

screen.create\_rectangle( x, y, x+150, y+150, fill =checkerSquareColour )