COMP 204

Functions

Yue Li
based on material from Mathieu Blanchette and Carlos Oliver Gonzalez

Functions: Why we need them

In large programs, we often need to perform several times the same type of computation. Examples:

- Ask the user for some input and check its validity
- Calculate the distance between two points in the plane
- Find the largest element in a list

Until now, the only way we have to do this is to duplicate and adapt code. This is bad because:

- It is very error-prone, hard to debug and maintain
- It makes the program unnecessarily large
- It makes the program hard to read

Example: you use the same distance equation in 10 different programs but later on decide to change the distance calculation.
Functions: Allow re-using a piece of code without duplicating it.
We've used many functions already: print(), sqrt(), isdecimal()..
Today, we learn how to define *our own* functions.

Functions: the first example

```
1  # This is the printWelcome function
2  def printWelcome():
3      print("**************************")
4      print("* Welcome to COMP 204! *")
5      print("**************************")
6
7  # This is now outside the printWelcome function
8  printWelcome()
9  print("My name is Yue")
10 # Some more code
11
12 #print again
13 printWelcome()
14 print("etc...")
15 #and again
16 printWelcome()
```

Notes:

- Use the keyword `def` to define our own functions.
- Once the function is defined, just call it using its name and its code will execute.
- **Note:** without a call, the function's code will not be executed.

The anatomy of a function

```
# function header
def function_name( function_arguments ):
    # body of function
    # ...
    # ...

# rest of program
```

- Function header
  1. **def** tells Python you are defining a function
  2. **function_name**. Functions are objects so we give them names
  3. **(function_arguments)** Objects you would like the function to work on (optional)
- Function body
  - Any code that is tabbed at least once and follows the **header** is stored in the function.

Functions with arguments

Without arguments, a function always executes the same thing.
For more flexibility, we pass arguments to the function.

```
1  # This function welcomes a student to COMP 204
2  def printWelcome204(studentName):
3      print("Dear",studentName)
4      print("Welcome to COMP 204")
5
6  # This function welcomes a student to any course
7  def printWelcome(studentName, courseName):
8      print("Dear",studentName)
9      print("Welcome to", courseName)
10
11
12 # This is now outside the printWelcome function
13 printWelcome204("Yang")
14 printWelcome204("Alessandro")
15 printWelcome("Veronica", "COMP 204 Winter 2019")
```

What happens when a function is called?

When a function is called:

- A new *local* variable is created for each argument (if any)
- The value of each argument variable is initialized to that provided with the function call
- The body of the function is executed. This may include defining/using other local variables.
- When the body is finished executing,
  - We discard local variables
  - We go back to the line where the function was called, and continue execution from there.

Note: A function can call another function. For example: the printWelcome() function calls the print() function.

The `return` statement

Until now, our functions print text, but the result of their computation cannot be communicated to the rest of the program.

- The return statement is a special word that lets the function "emit' an object i.e. output.
- This is useful because it lets the person who called the function store the output in memory and perform operations with it later on.
- **return** is NOT the same as print()
- When Python reaches a return statement it automatically *exits* the function.

Example 2: Computing Euclidean distance

```
1  import math
2
3  # this function calculates the distance between
4  # two points (x1, y1) and (x2, y2) in Euclidean space
5  def distance(x1, y1, x2, y2):
6      d = math.sqrt( (x1-x2)**2 + (y1-y2)**2 )
7      return d
8      print("Hello") #this is  never reached
9
10 myDistance = distance(3,1,5,7)
11 print("The distance is", myDistance)
12
13 print("The distance is ", distance(3,1, 5,7) )
14
15 print(d) # error: d is not accessible
16         # outside the distance function
```

```
1  import math
2
3  # this does the same without saving the local variable
4  def distance(x1, y1, x2, y2):
5      return math.sqrt( (x1-x2)**2 + (y1-y2)**2 )
6
7  print("The distance is ", distance(3,1, 5,7))
```

Functions: Why we need them

Functions are useful because they enable :

- **Code re-use:**
  - Once you've written a function *and made sure it works*, you can re-use it as many times as needed, from any program you want.
  - You can also re-use code written by others
  - Other can re-use you code
- **Encapsulation:**
  - As the user of a function, all you need to know is its name, arguments, and what it outputs. No need to worry about it works.
  - Allows breaking down complex tasks into small, easy to understand subtasks
  - Allows thinking about a problem at a high-level, focussing on the aspects that matter to your project.

Example 3: Re-visit the nuclear example

```
1  import math  # this imports the math module
2
3  def euclid(xHome, yHome, xAcc, yAcc):
4      return math.sqrt((xHome - xAcc)**2 + (yHome - yAcc)**2)
5
6  def evaluateRisk(distance):
7      if distance <= 20:
8          print("You must evacuate")
9      elif distance <= 40:
10         if (input("Are you pregnant? (yes/no) ") in ["yes","
       Yes","Y","y"]):
11             print("You must evacuate")
12         else:
13             print("Evacuation is recommended")
14     else:
15         print("No need to evacuate")
16
17 def evacuateAssessmentMain():
18     xAcc = float(input("Enter x coord. of nuclear: "))
19     yAcc = float(input("Enter y coord. of nuclear: "))
20     xHome = float(input("Enter x coordinate of home: "))
21     yHome = float(input("Enter y coordinate of home: "))
22     evaluateRisk(float(euclid(xHome, yHome, xAcc, yAcc)))
23
24 evacuateAssessmentMain()
```

Example 3: Safe input for integers

Goal: Write a function that repeatedly asks a user to enter an integer, until the number entered in within a desired range. Once a valid input has been entered, return that value.

```
# Asks user to enter a value by printing message
# Repeats until value is between minVal and maxVal
def inputInRange(message, minVal, maxVal):

    while True: # loops until return statement is executed
        n = int(input(message))
        if n >= minVal and n <= maxVal:
            return n
        else:
            print("Number outside of range",minVal, maxVal)

age = inputInRange("Enter age: ",0,150)
height = inputInRange("Enter height (in cm): ",0,250)
```

Example 4: Safe input for strings

Goal: Write a function that repeatedly asks a user to enter a string, until the number entered in within a desired lis of acceptable values. Once a valid input has been entered, return that value.

```
1  # Asks user to enter a string value by printing message
2  # Repeats until value is within list acceptable values
3  def inputInList(message, acceptableList):
4
5      while True: # loops until return statement is executed
6          s = input(message)
7          if s in acceptableList: # tests if s is in list
8              return s
9          else:
10             print("Please respond by ",acceptableList)
11
12
13 history = inputInList("History of diabetes? ", ["yes","no"])
14 gender = inputInList("Gender? ", ["female","male"])
```

Example 5: Checking prime number

- A function body can have multiple return statements. The first one encountered during execution will end the function
- Exercise: write a function that returns True if it is given a prime number and False otherwise.

```
# This function return True if the integer
# provided as argument is a prime number
def isPrime( n ):
    # look at all candidate factors of n
    for f in range(2, n):
        # see if f is a factor of n
        # by computing the remainder of the division
        if n % f == 0:
            return False
    return True

if isPrime(int(input("Enter a number: "))):
    print("The number is prime")
else:
    print("The number is not prime")
```

Example (advanced): Recursion: function that calls itself

```
1 # a function that calls itself
2 def countDownRecursion(count):
3     if count > 0:
4         print(count)
5         countDownRecursion(count-1)
6
7 countDownRecursion(10)
```