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

**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: A first example

```python
# This is the printWelcome function
def printWelcome():
    print("********************************")
    print("* Welcome to COMP 204! *")
    print("********************************")

# This is now outside the printWelcome function
printWelcome()
print("My name is Mathieu")
# Some more code

# print again
printWelcome()
print("etc...")
# and again
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

```python
# 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.

```python
# This function welcomes a student to COMP 204
def printWelcome204(studentName):
    print("Dear", studentName)
    print("Welcome to COMP 204")

# This function welcomes a student to any course
def printWelcome(studentName, courseName):
    print("Dear", studentName)
    print("Welcome to", courseName)

# This is now outside the printWelcome function
printWelcome204("Yang")
printWelcome204("Alessandro")
printWelcome("Veronica", 364)
```
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 “spit out” 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: Computing Euclidean distance

```python
import math

# this function calculates the distance between
# two points (x1, y1) and (x2, y2) in Euclidean space
def distance(x1, y1, x2, y2):
    d = math.sqrt((x1-x2)**2 + (y1-y2)**2)
    return d

print("Hellooo") #this is never reached

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, focusing on the aspects that matter to your project.
Example: Checking prime number

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

```python
# 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

num = int(input("Enter a number: "))
if isPrime(num):
    print("The number is prime")
else:
    print("The number is not prime")
```
Goal: Write a function that repeatedly asks a user to enter an integer, until the number entered is within a desired range. Once a valid input has been entered, return that value.

```python
# 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)
```
Goal: Write a function that repeatedly asks a user to enter a string, until the number entered is within a desired list of acceptable values. Once a valid input has been entered, return that value.

```python
# Asks user to enter a string value by printing message
# Repeats until value is within list acceptable values
def inputInList(message, acceptableList):

    while True:  # loops until return statement is executed
        s = input(message)
        if s in acceptableList:  # tests if s is in list
            return s
        else:
            print("Please respond by ", acceptableList)

history = inputInList("History of diabetes? ", ["yes", "no"])
gender = inputInList("Gender? ", ["female", "male"])
```
Example: Hydrophobic patches

- Protein sequences are made of amino acids.
- Some amino acids (G, A, V, L, I, P, F, M, W) are hydrophobic (i.e. they don’t like to interact with water molecules).
- Some proteins contain hydrophobic patches, which are portions of the sequence that start and end with an hydrophobic amino acid and where at least 80% of the amino acid are hydrophobic.
- For example, in the sequence EDAYQIALEGAASTE, the longest hydrophobic patch is IALEGAA.

**Goal:** Write a function that identifies the longest hydrophobic patch in a given protein sequence.

**Solution:** See hydrophobicPatch.py