COMP 204
Functions II

Yue Li
based on material from Mathieu Blanchette and Carlos Oliver Gonzalez
Quiz 12 password
Getting help

- TAs and I are available to help, and not just for assignments!
- The Computer Science Undergraduate Student (CSUS) association has a help desk where you can drop in with questions any time 10am-5pm in Trottier 3090.
Quiz 11 review: order the functions get executed

```python
import math  # this imports the math module

def euclidean(xHome, yHome, xAcc, yAcc):
    return math.sqrt((xHome - xAcc)**2 + (yHome - yAcc)**2)

def pregnantQuestion():
    if (input("Are you pregnant? (yes/no) ") == "yes"):
        print("You must evacuate")
    else:
        print("Evacuation is recommended")

def evaluateRisk(distance):
    if distance <= 20:
        print("You must evacuate")
    elif distance <= 40:
        pregnantQuestion()
    else:
        print("No need to evacuate")

def evacuateAssessmentMain():
    xAcc = 20; yAcc = 30
    xHome = 40; yHome = 50
    evaluateRisk(float(euclidean(xHome, yHome, xAcc, yAcc)))

evacuateAssessmentMain()
```
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 EDAYQIALEGAAASTE, the longest hydrophobic patch is IALEGAA.

**Goal:** Write a function that identifies the *longest* hydrophobic patch in a given protein sequence.
Find longest hydrophobic patch by *divide-and-conquer*

```python
findLongestHydrophobicPatch(protein)
```

```python
isHydrophobicPatch(sequence)?
```

```python
EDAYQIALEGAASTE
```

- **outer for loop:**
  - start position from `start = 0`

- **inner for loop:**
  - end position from `end = start + 1`

```python
isHydrophobicPatch(sequence)?
```

```python
isHydrophobic('E')
```

# (1) first a.a.

```python
isHydrophobic('L')
```

# (2) last a.a.

```python
isHydrophobic(s[aa])
```

# (3) length of hydrophobic amino acids (min 80%)

```python
patchLen += isHydrophobic(s[aa])
```

Not the most efficient way (discussed a bit later)
Example: Hydrophobic patches

Divide-and-Conquer (bottom up approach): Break it down into small, manageable tasks and start with the lowest tasks

1. Write a function that checks if a given amino acid is hydrophobic
2. Write a function that checks if a given sequence is a hydrophobic patch:
   ▶ Starts and ends with a hydrophobic amino acid
   ▶ Made at 80% or more of amino acids (i.e. count hydrophobic amino acids; see if count is at least 0.8*length)
3. Use nested for or while loop to iterate over all possible start and end points of a candidate patch. Use function above to test if it is a patch. If it is, calculate length and update the variable that keeps track of the longest patch found so far.
4. Report longest patch found
isHydrophobic function

```python
# This function returns True if aa is a hydrophobic amino acid
def isHydrophobic(aa):
    hydrophobic = ["G","A","V","L","I","P","F","M","W"]
    # This checks if aa is equal to an object in the list
    if aa in hydrophobic:
        return True
    else:
        return False

# This is a shorter way to do the same thing
def isHydrophobic2(aa):
    return (aa in ["G","A","V","L","I","P","F","M","W"])
```
isHydrophobicPatch function

```python
# This function tests whether a given sequence contains at least 80% of hydrophobic amino acids
def isHydrophobicPatch(sequence):
    # test if sequence starts and ends with a hydrophobic aa
    # If not, it is not a hydrophobic patch, so return False
    if isHydrophobic(sequence[0]) == False or isHydrophobic(sequence[-1]) == False:
        return False

    # Count the fraction of hydrophobic amino acids
    hydrophobicCount = 0
    for aa in sequence:
        if isHydrophobic(aa):
            hydrophobicCount += 1

    # See if we have enough hydrophobic amino acids
    if hydrophobicCount >= 0.8 * len(sequence):
        return True
    else:
        return False

# shorter way to do the same with one boolean expression

def isHydrophobicPatch2(sequence):
    return isHydrophobic(sequence[0]) and \
    isHydrophobic(sequence[-1]) and \
    len([aa for aa in sequence if isHydrophobic(aa)]) > 0.8*len(sequence)
```
findLongestHydrophobicPatch function

```python
# This returns the longest hydrophobic patch found in a sequence

def findLongestHydrophobicPatch(protein):
    longestPatch=""  # the longest patch found so far

    # for every possible starting point
    for start in range(0, len(protein)):
        
        # and every possible end point
        for end in range(start +1, len(protein)+1):
            # get the sequence
            candidate = protein[start:end]

            # test hydrophobicity
            if isHydrophobicPatch(candidate):
                # if longer than longest seen so far, update
                if len(candidate)>len(longestPatch):
                    longestPatch = candidate

    return longestPatch
```

This is an exhaustive search and not the most efficient algorithm. How do we improve it? How much can we improve?
Recursion version (advanced):
findLongestHydrophobicPatch_recur

```python
def findLongestHydrophobicPatch_recur(protein, start, end):
    if start < end and end <= len(protein):
        if isHydrophobicPatch(protein[start:end]):
            return protein[start:end]
        else:
            patch1 = findLongestHydrophobicPatch_recur(protein, start+1, end)
            patch2 = findLongestHydrophobicPatch_recur(protein, start, end-1)
            else:
                return ""

    if len(patch1) > len(patch2):
        return patch1
    else:
        return patch2

# code to test our function
protein = input("Enter protein sequence: ")
patch = findLongestHydrophobicPatch_recur(protein, 0, len(protein))
print("Longest hydrophobic patch is ", patch)
```
Positional arguments

The functions we have seen so far take as input *positional arguments*. Arguments are passed in the same order as the function definition.

Example:

```python
def inputInRange(message, minVal, maxVal):
```

Notes:

- Every call to the function *must* provide exactly three objects as arguments.
- The order of the arguments matter:
  
  ```python
  inputInRange("Enter age", 0, 150)
  ```
  
  is not the same thing as
  
  ```python
  inputInRange("Enter age", 150, 0)
  ```
Optional arguments

Another way to pass arguments to functions is to use *keyword arguments*. Example:

```python
# The function takes two keyword arguments
def inputInRange(message, minVal = 0, maxVal = 100):
    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:")
height = inputInRange("Enter height (in cm):", maxVal = 250)
weight = inputInRange("Enter weight:", maxVal = 250, minVal = 20)
```

Notes:

- Keyword arguments are optional when calling the function. If the caller does not provide them, they are set to their default value specified in the function header.
- Keyword arguments must come *after* positional arguments.
- Keyword arguments can be specified in any order.
- Useful when a function can take a large number of optional arguments.
Returning multiple outputs

A function can only return one object. What if a function needs to return multiple pieces of information? Idea: The object returned can be a compound object (list, tuple).

```python
# This returns a tuple made of the longest hydrophobic patch found in a sequence, along with its start and end positions

def findLongestHydrophobicPatch(protein):
    longestPatch=""
    for start in range(0, len(protein)):
        for end in range(start + 1, len(protein)):
            candidate = protein[start:end]
            if isHydrophobicPatch(candidate):
                if len(candidate)>len(longestPatch):
                    longestPatch = candidate
                    longestPatchStart = start
                    longestPatchEnd = end

    # this returns a tuple
    return (longestPatch, longestPatchStart, longestPatchEnd)

# code to test our function
protein = input("Enter protein sequence: ")
patch, s, e = findLongestHydrophobicPatch(protein)
print("Longest hydrophobic patch is ",patch)
print("It goes from position" ,s,"to position" ,e)
```
Recursion version 2:

```python
def findLongestHydrophobicPatch_recur2(protein, start, end):
    if start < end and end <= len(protein):
        if isHydrophobicPatch(protein[start:end]):
            return (protein[start:end], start, end)
        else:
            patch1, patch1_start, patch1_end =
            findLongestHydrophobicPatch_recur2(protein, start + 1, end)
            patch2, patch2_start, patch2_end =
            findLongestHydrophobicPatch_recur2(protein, start, end - 1)
    else:
        return (""", 0, 0)

    if len(patch1) > len(patch2):
        return patch1, patch1_start, patch1_end
    else:
        return patch2, patch2_start, patch2_end
```
The scope of variables

When inside a function, the only variables that are available are:

▶ Local variables: The function’s arguments, and all the variables defined within that function.
  ▶ When we return from a function, all local variables are discarded.
  ▶ It is possible for a function to have a local variable called x, even if a global variable x already exists. Those are considered two different variables, and only the local version is used.

▶ Global variables: Those defined outside any function. Their value can be accessed within a function, but not changed.

Notes:

▶ Avoid referring to global variables within functions. It makes code very confusing.

▶ It is actually possible for a function to change the value of global variables, but this is rarely a good thing to do, so we will not explain it here.
```python
def fun1():
    x=53  # is local to fun1
    print("Within fun1, x = ",x)

def fun2(x):
    x=2  # is local to fun2
    print("Within fun2, x = ",x)

def fun3():  # x is not defined within fun3, 
    # so we use the global variable
    print("Within fun3, x = ",x)

x=17
print("To start, x = ",x)
fun1()
print("After fun1, x = ",x)
fun2(x)
print("After fun2, x = ",x)
fun3()
print("After fun3, x = ",x)
```

Output:
To start, x = 17
Within fun1, x = 53
After fun1, x = 17
Within fun2, x = 2
After fun2, x = 17
Within fun3, x = 17
After fun3, x = 17