COMP 204
Functions II

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

Solution: See hydrophobicPatch.py
Example: Hydrophobic patches

Approach: Break it down into small, manageable tasks

- Write a function that checks if a given amino acid is hydrophobic
- 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)
- 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.
- Report longest patch found
# This function returns True if aa is a hydrophobic amino acid

def isHydrophobic(aa):

    # This checks if aa is equal to an object in the list hydrophobic
    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 hydrophobic)
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
A more compact `isHydrophobicPatch` function

```python
# This is a shorter way to do the same thing
def isHydrophobicPatch2(sequence):
    if not isHydrophobic(sequence[0]) or not isHydrophobic(sequence[-1]):
        return False

    # This is called a list comprehension; we will introduce it later
    hydrophobicCount = len([aa for aa in sequence if isHydrophobic(aa)])
    return hydrophobicCount >= 0.8*len(sequence)
```
The `findLongestHydrophobicPatch` function

```python
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

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

The functions we have seen so far take as input *positional arguments*. 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: 
  `inputInRange("Enter age", 0, 150)` is not the same thing as 
  `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 parameters.
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)
```
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