COMP 204

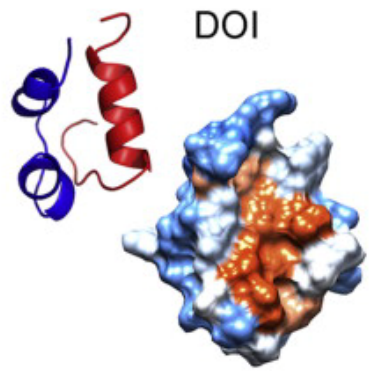
Functions II

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

Quiz 11 password

Example: Hydrophobic patches

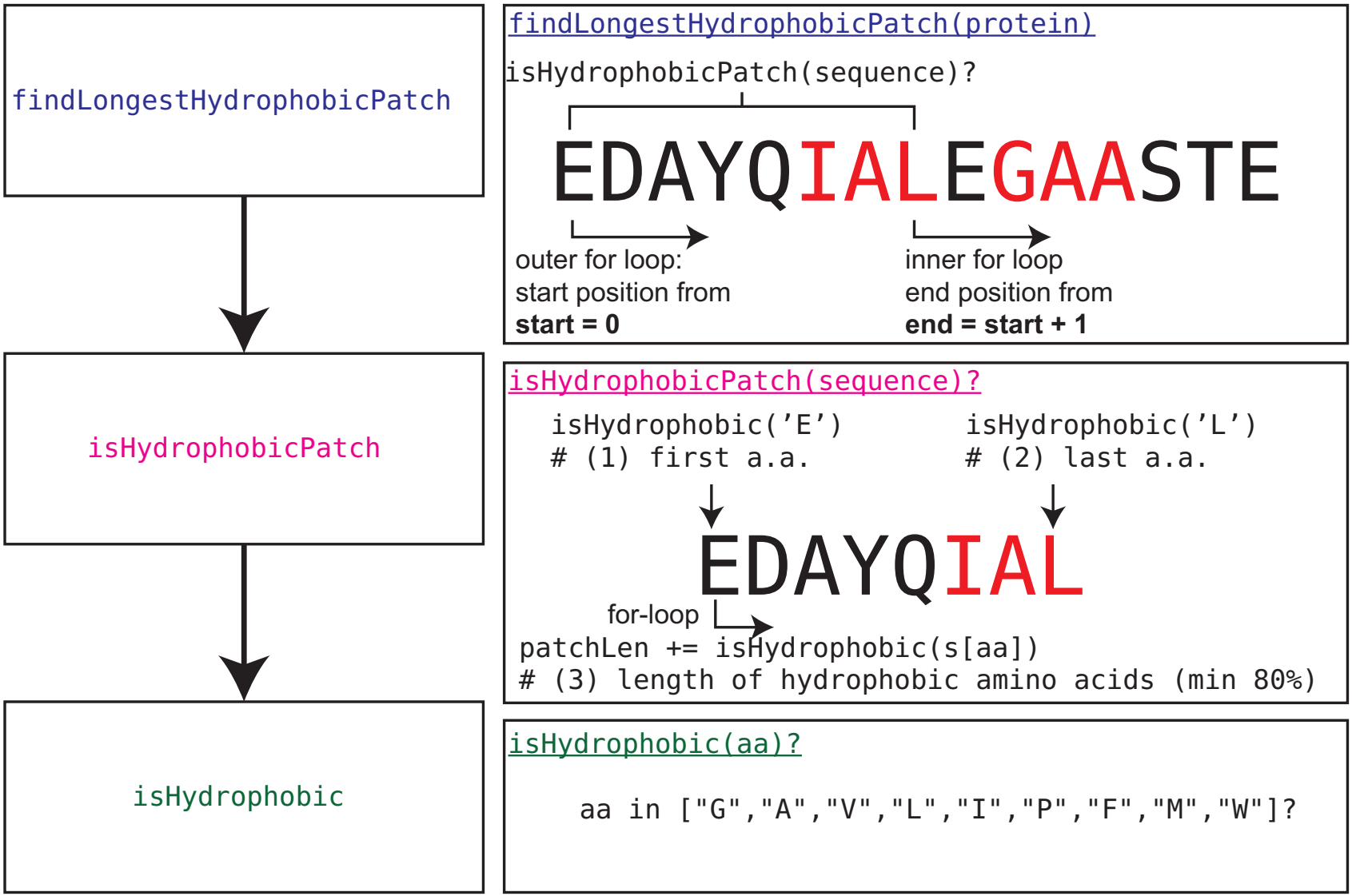
- Protein sequences are made of amino acids.
- Some amino acids (G, A, V, L, I, P, F, M, W) are hydrophobic (i.e. they don't like to interact with water molecules).
- Some proteins contain *hydrophobic patches*, which are portions of the sequence that start and end with an hydrophobic amino acid and 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.

Find longest hydrophobic patch by *divide-and-conquer*



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

```
1  # This function returns True if aa is a hydrophobic amino
       acid
2  def is_hydrophobic(aa):
3      hydrophobic = ["G","A","V","_I","I","_p","F","M","W"]
4
5      # This checks if aa is equal to an object in the list
       hydrophobic
6      if aa in hydrophobic:
7          return True
8      else:
9          return False
10
11 # This is a shorter way to do the same thing
12 def is_hydrophobic2(aa):
13     return (aa in ["G","A","V","_I","I","_p","F","M","W"])
```

isHydrophobicPatch function

```
# This function tests whether a given sequence
# contains at least 80% of hydrophobic amino acids
def is_hydrophobic_patch(sequence):
    # test if sequence starts and ends with a hydrophobic aa
    # If not, it is not a hydrophobic patch, so return False
    if is_hydrophobic(sequence[0]) == False or
    is_hydrophobic(sequence[-1]) == False:
        return False
    # Count the fraction of hydrophobic amino acids
    hydrophobicCount = 0
    for aa in sequence:
        if is_hydrophobic(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 is_hydrophobic_patch2(sequence):
    return is_hydrophobic(sequence[0]) and \
           is_hydrophobic(sequence[-1]) and \
        len([aa for aa in sequence if is_hydrophobic(aa)]) >
     0.8*len(sequence)
```

findLongestHydrophobicPatch function

```
# This returns the longest hydrophobic patch found in a sequence
def find_longest_hydrophobic_patch(protein):
    longest_patch=""  # 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 is_hydrophobic_patch(candidate):

                # if longer than longest seen so far, update
                if len(candidate)>len(longest_patch):
                    longest_patch = candidate

    return longest_patch
```

This is an exhaustive search and not the most efficient algorithm. How do we improve it? How much can we improve?

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:

```
1 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:

```
# 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 heigth (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

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).

```
1  # This returns a tuple made of the longest hydrophobic patch
2  # found in a sequence, along with its start and end
       positions
3  def findLongestHydrophobicPatch(protein):
4      longestPatch=""
5      for start in range(0,len(protein)):
6          for end in range(start+1,len(protein)):
7              candidate = protein[start:end]
8              if isHydrophobicPatch(candidate):
9                  if len(candidate)>len(longestPatch):
10                     longestPatch = candidate
11                     longestPatchStart = start
12                     longestPatchEnd = end
13     # this returns a tuple
14     return (longestPatch,longestPatchStart,longestPatchEnd)
15
16 # code to test our function
17 protein = input("Enter protein sequence: ")
18 patch, s, e = findLongestHydrophobicPatch(protein)
19 print("Longest hydrophobic patch is ",patch)
20 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.

```
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