CSE 503

Introduction to Computer Science for Non-Majors

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Day 12
Gaining Execution Insight

Announcements

- Lab #1 due tonight!
- Lab #2 released, due 10/10/22 @ 11:59PM
  - Autolab will open for submission next Monday.
  - Get in the habit of writing tests on your own.

Recap

- Review of for loops in JavaScript
- Introduction to a few more list/array operations
- The accumulation pattern:

```
let s = 0;          // Set an accumulator variable
for (x in seq) {    // Loop
  s = s + x;        // Accumulate into our variable
}
return s;         // Do something with the result
```

Exercise 1: Explode

Write a function called **`explode`**, which takes a string and returns a list (Python) or an array (JS) where each element is a character of the string.

For example **`explode("Hello")`** must return **`["H", "e", "l", "l", "o"]`**.

Remember the new list/array operations we saw last time.

Does this match the accumulation pattern we saw last time?

If so, what is our operation, and does it have an identity?

Exercise 2: Explode More

Write a function called **`explodeMore`** that takes a list/array of strings, and explodes them all into a single list/array of characters.

For example, **`exploreMore(["Hi", "Bye"])`** must return **`["H", "i", "B", "y", "e"]`**.

Gaining Insight into Code Execution

- As you start writing bigger programs, they become more complex
- Things **WILL** go wrong
  - A good programmer doesn't program perfectly every time…
  - …but they know how to fix things when they go wrong.
- The first step into fixing a problem is diagnosing the problem
- To do so, you need to understand what your program is actually doing

How can we gain this insight using stuff we have already seen?

Instrumenting our Previous Example

```
1.  def explode(s):
2.  l = []
3.  print("Input String: " + s)
4.  print("Starting accumulator value: " + str(l))
5.
6.  for c in s:
7.    print("Starting a loop iteration with c = " + c)
8.    l.append(c)
9.    print("At the end of the iteration, a is " + str(l))
10.
11. print("After the loop, a is " + str(l))
12. return l
```

Print inputs and/or initial values

Print relevant values in each iteration

Print out the final result

Now let's revisit explodeMore...

DNA Examples

DNA consists of chains of nucleotide bases, adenine (A), cytosine (C), guanine (G), and thymine (T).

Nucleotide sequences can be represented by a string of A, C, G, and T characters.

Let's try a couple of examples based on this principle…

DNA Count Example

Write a function called **`dnaCount`** that accepts a DNA string, and a string representing a single nucleotide base ("A", "C", "G", or "T"). The function should return the number of times that base appears in the DNA string.

For example:

`dnaCount("ACAGCCTAAG", "A")` must return **`4`**

`dnaCount("ACAGCCTAAG", "G")` must return **`2`**

DNA Similarity Example

Write a function called **`dnaSimilarity`** that takes two DNA sequences as strings, and returns a percentage (from 0.0 to 1.0) based on how similar the two strings are. You can compute it by taking the number of matching bases and divide by the total number of bases. You can assume the DNA strings passed in are of the same length.

For example:

`dnaSimilarity("ACAGCCTAAG", "ACAGCCTAAG")` would evaluate to **`1.0`**

`dnaSimilarity("ACAGCCTAAG", "ACAGCGGTCC")` would evaluate to **`0.5`**

DNA Frequency Example

Write a function called **dnaFrequency** that takes a single DNA string, and returns a list of 4 lists, one for each base and its count.

For example:

`dnaFrequency("ACAGCCTAAG")` must return
`[["A", 4],["C", 3], ["G",2],["T",1]]`

`dnaFrequency("TCAGCCTAAG")` must return
`[["A", 3],["C", 3], ["G",2],["T",2]]`