CS1210 Lecture 8 Sep. 10, 2021

- HW2 due next Tuesday
  - Note: Assignment says “You may not use any string methods except for lower()”. But the len() function and “in” operator are allowed. It is not a string method.
- DS3 will be posted today, due Tuesday 8pm. Basic loop and string practice. Attendance again not required but TAs will this time spend about 10 minutes walking through the beginning examples in the ds3.py assignment
- Quiz 1 next Wednesday in class
- HW1 will be graded by Sunday night

Last time

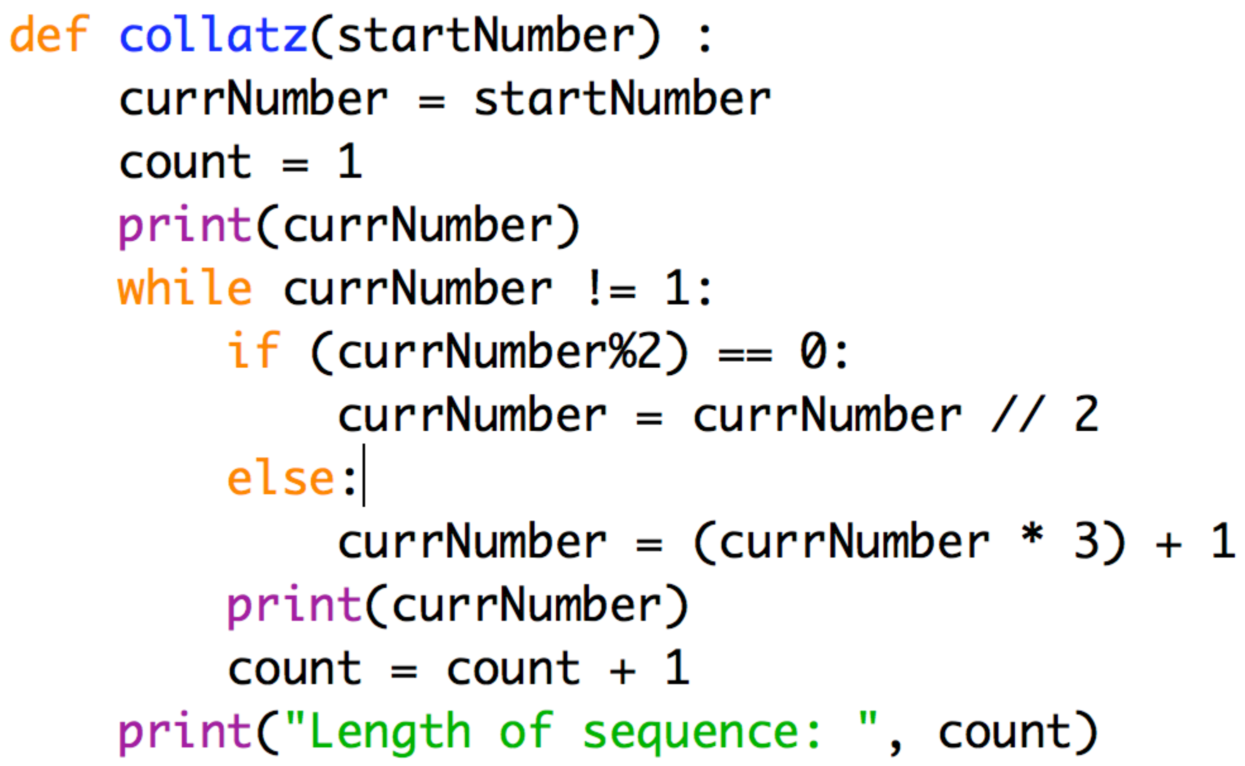
- Strings (Ch 9) and while loop examples

Today

- More while loops
  - collatz example. Can we always know whether loops terminate or not?
  - charExamples.py, relevant to HW2 Q3
  - **for** loops, an alternative way to iterate

While loop termination again Ch 8.5

- We (usually, but not always) want loops to terminate. People sometimes work to formally prove that a loop terminates
- *But sometimes we're not sure*!



```
def collatz(startNumber) :
    currNumber = startNumber
    count = 1
    print(currNumber)
    while currNumber != 1:
        if (currNumber%2) == 0:
            currNumber = currNumber // 2
        else:
            currNumber = (currNumber * 3) + 1
        print(currNumber)
        count = count + 1
    print("Length of sequence: ", count)
```

- This is known as the Collatz or 3n+1 problem (https://en.wikipedia.org/wiki/Collatz_conjecture) No one has been able to prove that this will terminate for all positive n!   collatz.py

A looping example helpful for Q3 of HW2

- Q3 asks you to write a function that takes four inputs:
  - a string, inputString, of lower case characters
  - a string, minLetter
  - a string, lettersToIgnore
  - a string, specialLetter

  and returns 1) the "smallest" character in inputString that is both greater than minLetter and not in lettersToIgnore, 2) the highest index at which that letter occurs, and 3) True if specialLetter occurs an odd number of times in inputString

- Study the three functions in charExamples.py to help you get started. They use simple loops and no string operations other than "in". It's good to practice this kind of loop pattern: iterating through a string maintaining one or more variables related to a "best" or "minimum" or "largest"

We seen looping over strings with **while**

Using [] and len, you can write while loops that do things with each character in a string:

```
currentIndex = 0
while currentIndex < len(myString):
    currentChar = myString[currentIndex]
    ….
    …. (loop body – typically does something
    …. with character, currentChar)
    ….
    currentIndex = currentIndex + 1
```

We'll continue to practice this over the next couple of weeks; it is very important that you understand this general pattern of stepping through a string (or, later, other sequence) by using a loop and incrementing an index

for loops and strings

Python provides an alternative, more concise way to iterate over strings

```
for currentChar in myString:
    ….
    …. (loop body – typically does something
    …. with character, currentChar)
    ….
```

The body of the **for** loop gets executed once for each character in the string, myString. On the first iteration, currentChar is bound to the first (i.e. index 0) character of myString. On the second iteration, currentChar is bound to the second (i.e. index 1) character, etc. This loop and the one on the previous page are equivalent! *You need to be able to convert* ***for*** *loops to equivalent* ***while*** *loop! It's a simple "robotic" process (and I almost always test this on the second quiz)*

lec8loopchars.py

Looping on strings with **for**

```
def findChar(charToFind, stringToSearch):
    for char in stringToSearch:          lec8findChar.py
        if char == charToFind:
            print(‘found it’)
            return      <— leaves function immediately
```

But what if specification is instead to return the index of character if found, and length of given string if not?

Looping on strings with **for**

```
def findChar(charToFind, stringToSearch):
    indexOfCharSought = len(stringToSearch)          lec8findChar.py
    currentIndex = 0
    for char in stringToSearch:
        if char == charToFind:
            indexOfCharSought = currentIndex
                                                <— exits loop immediately*
        currentIndex = currentIndex + 1
    return indexOfCharSought
```

What is different if we remove break Is the result different/still correct?

** Be careful; if not used well **break** can yield confusing code*

Ch 9.13: string **in** operator

- ‘a’ in myString returns True if ‘a’ is in myString, False otherwise

Write function inBoth(string1, string2) that prints all characters that appear in both:

```
def inBoth(string1, string2):
    for c in string1:
        if c in string2:
            print(c)
```

Demo/exercises

- lec8exercises.py debugging exercises involving **for** and strings

Next time: printFirstNPrimes

- A prime number is an integer greater than one that has no divisors other than 1 and itself.
  - 2, 3, 5, etc.
- Goal: implement function printFirstNPrimes(n) that takes integer n as input and prints the first n prime numbers.

```
>>> printFirstNPrimes(4)
2
3
5
7
```

Top-down design of printFirstNPrimes

Express algorithm in comments, like an outline.
Incrementally refine and implement steps.

```
def printFirstNPrimes(n):
    # starting at 2, count upwards, testing
    #     candidate integers for primeness,
    #     printing those that are prime
    #     and stopping after n
    #     have been printed
```

Next time

- Development of printFirstNPrimes
- Review and examples for Quiz 1