CS2110 Lecture 8 Feb. 10, 2021

- HW2 due tomorrow night
  - Note: Q2 says “no string methods” are allowed. But the len() function is allowed. It is not a string method.
- Quiz 1 on Friday in class

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
  - Development of printFirstNPrimes

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 three inputs:
  - a string, inputString, of lower case characters
  - A string, minLetter
  - A string, lettersToIgnore

  and returns 1) the "smallest" character in inputString that is both greater than minLetter and not in lettersToIgnore, and 2) the highest index at which that letter occurs

- 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"

lec7primes.py : 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
```

Top-down design of printFirstNPrimes

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

```
def printFirstNPrimes(n):
    candidate = 2
    while (numPrimesPrinted != n):
        # test candidate for primeness
        # print, update numPrimesPrinted if prime
        candidate = candidate + 1
```

Top-down design of printFirstNPrimes

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

```
def printFirstNPrimes(n):
    candidate = 2
    numPrimesPrinted = 0
    while (numPrimesPrinted != n):
        # test candidate for primeness
        # print, update numPrimesPrinted if prime
        candidate = candidate + 1
```

Top-down design of printFirstNPrimes

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

```
def printFirstNPrimes(n):
    candidate = 2
    numPrimesPrinted = 0
    while (numPrimesPrinted != n):
        isPrime = numIsPrime(candidate)
        # print, update numPrimesPrinted if prime
        candidate = candidate + 1
```

Top-down design of printFirstNPrimes

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

```
def printFirstNPrimes(n):
    candidate = 2
    numPrimesPrinted = 0
    while (numPrimesPrinted != n):
        isPrime = numIsPrime(candidate)
        if isPrime:
            print(candidate)
            numPrimesPrinted = numPrimesPrinted + 1
        candidate = candidate + 1
```

```
# stub like this VERY
# USEFUL for testing!!
#
def numIsPrime(n):
    isPrime = True
    return isPrime
```

Now, just need to implement numIsPrime() BUT first test this code using "stub" numIsPrime() !

lec7primes.py

Next: finish printFirstNPrimes by replacing stub isNumPrime with correct code

Again, develop isNumPrime in top-down fashion:

```
def isNumPrime(n):
    # presume number is prime
    # check potential divisors 2 .. n-1. If any evenly divides n
    # then n is not prime
```

Top-down design of printFirstNPrimes

Now develop isNumPrime in similar fashion:

```
def isNumPrime(n):
    isPrime = True
    # check potential divisors 2 .. n-1. If any evenly divides n
    # then n is not prime
```

Top-down design of printFirstNPrimes

Now develop isNumPrime in similar fashion:

```
def isNumPrime(n):
    # presume number is prime
    isPrime = True
    # check potential divisors 2 .. n-1. If any evenly divides n
    potentialDivisor = 2
    while potentialDivisor < n:
        # check if potential divisor evenly divides n,
        #     update isPrime if it does
        potentialDivisor = potentialDivisor + 1

    return isPrime
```

Top-down design of printFirstNPrimes

Now develop isNumPrime in similar fashion:

```
def isNumPrime(n):
    # presume number is prime
    isPrime = True
    # check potential divisors 2 .. n-1. If any evenly divides n
    potentialDivisor = 2
    while potentialDivisor < n:
        # check if potential divisor evenly divides n,
        #     updating isPrime if it does
        if (n % potentialDivisor) == 0:
            isPrime = False
        potentialDivisor = potentialDivisor + 1

    return isPrime
```

lec7primes.py

Note: this can be improved:

1) When find divisor, stop searching, return False
2) Search doesn't need to go to n-1. Can stop when potential divisor reaches square root of n (if n has divisor bigger than its square root, it must also have one smaller)

BUT general rule: worry about correctness before working on optimizations like this

Next time

Quiz 1, in class