CS1210 Lecture 9 Sept. 13, 2021

- Quiz 1: Wednesday in class
- HW1 scores have been posted
- HW2 due 8am tomorrow
  - Don’t violate the Academic Honesty Policy on the course website. Some borderline violations/collaborations in HW1 that will be watched further.
- DS3 due 8pm tomorrow
  - Again, optional to attend BUT this time TA will walk students through some of the example

Today

- A debugging example
- Introduce the range() function
- Develop printFirstNPrimes
- Discuss quiz

HW1 Most Common Comments

Q1
-1 doesn’t correctly handle trip lengths that are exact multiples of 8. E.g. you say an 8 hour trip should have 1 hotel night
-1 hotel nights calculation is not correct
-1 lunches calculation is not correct
-1 breakfasts calculation is incorrect
-1 dinner calculation not correct
-1 doesn’t handle rest days correctly (this is more specific than just generally having incorrect hotel nights)
-1 loops were not allowed. See the assignment spec. I talked in class about how to do this easily without loops

-5 file does not successfully load into Python due to basic syntax errors. As discussed carefully in class, we cannot give significant credit for files with syntax errors. Syntax errors are easy to correct, and must be eliminated before submitting your code.

Q2
-1 Q2 does not call Q1. The assignment spec said that it must.
-1 prints the required value but returns nothing. It needs to return the string as well

Note: it is completely unnecessary/wasteful to call computeTripData *twice* from Q2. Call it *once* - it gives back everything needed in one call.

A debugging example

```
def is_reverse(word1, word2):
    if len(word1) != len(word2):
        return False
    i = 0
    j = len(word2)

    while j > 0:
        if word1[i] != word2[j]:
            return False
        i = i + 1
        j = j - 1

    return True
```

is_reverse should return True if word1 is the reverse of word2.
I.e. is_reverse(“abc”, “cba”) should return True while is_reverse(“ab”, “ab”) should return False

Is code correct?

code in lec9.py

The **range** function

Python's **range** function is very useful. There is no one clear place in the text where it is presented. It is first mentioned in 4.7 of the Turtle chapter, and then used in examples in Ch 9 and 10.

The range function produces values of a **range** type

The range type is another sequence type, like **list** and **string**.

range(9) is a sequence of the integers 0, 1, …, 8

range(2,6) is sequence 2, 3, 4, 5

range(2,13,3) is sequence 2, 5, 8, 11

Since range is a sequence type, (most of) the standard sequence operations apply (not nicely specified anywhere in text – go to Python sequence docs on-line)

range – standard sequence ops

```
>>> 5 in range(9)
True
>>> 5 in range(2,10,2)
?
>>> len(range(2,10,2))
?
>>> myRange = range(2,20,2)
>>> myRange[3:6]
?
>>> range(5) + range(5)
?
```

Exercise: use range with for to easily create lec8's loopchars() fn without while

lec9primes.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() !

lec9primes.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
```

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

lec9primes.py

Next time: quiz 1

- 45 minutes, 4 questions
- Probably:
  - One piece of code in which you must replace logical expression with some if/elif/elses (see ex1a and ex1b in lec9.py)
  - Two small functions to implement
    - Surely involving basic loops. Style and level of difficulty of HW2 and DS3 problems.
  - Compare two or more functions and determine whether or not they produce the same result on all inputs (see f1a, f1b in lec9.py)