

CS1 Lecture 7

Feb. 1, 2017

- HW 2 due Monday, Feb. 6, 9:00am
 - #1. google: python documentation print function
 - #2 and #3. some of you know how to use lists, etc. Don't! Read assignment carefully. Point is to practice looping and counting/indexing.
 - #4. As problem states, some inputs will yield infinite loops. That's okay/correct. (But can be annoying for Wing IDE users.)
- Second survey due Friday night. The survey is in the ICON course for your particular discussion section, *not* the lecture. (Note: this is different than the first survey, which was posted on the lecture's ICON course)
- For additional practice, try auto-corrected Python exercises at codingbat.com

Last time

- Finished conditional execution part of Chapter 5 (nested conditionals and if-elif-else)
- Began Chapter 7, Iteration

Ch 7: Iteration – the **while** statement

- Many computations involve repetition, doing the same (or nearly the same) things repeatedly (perhaps a few times, perhaps billions of times)
- You could already write a program to, say, print out the first 1000 integers

```
def printFirstThousand():
```

```
    print(1)
```

```
    print(2)
```

```
    ...
```

```
    print(1000)
```

- But Python (and other languages) provide statements to conveniently describe and control repetitive computations.

Ch 7 – the while statement

The **while** statement provides a very general mechanism for describing repetitive tasks.

What happens?

```
...  
... (B1: code before while)  
...  
while boolean expression:  
    ...  
    ... (B2: code in while body)  
    ...  
...  
... (B3: code after while)  
...
```


1. Execute B1 code
2. Evaluate Boolean exp
3. If True, do
 - 3a. eval B2 code.
 - 3b. jump to step 2 again
- If False, ignore B2 code and simply continue with step 4
4. Execute B3 code

Today and Friday

- Quick intro to beginning of Ch 8: Strings and another iteration construct, **for**
 - We'll cover Ch 8 in more detail Friday but will introduce parts useful for HW2 questions 2 and 3
- More **while** examples

Intro to Ch8: Strings and for

- So far, we've used strings to print things and only discussed one string operator, +. There are many other string operators and functions
- You can get individual characters within a string via the [] operator:

```
>>> myString = "python"
>>> myString[0]
'p'
>>> myString[3]
'h'
```
- The expression inside the brackets is called an **index**
 - In Python the first element has index 0 !!!  **IMPORTANT**
- And len() provides the number of characters

```
>>> len(myString)
6
```

Ch 8: string methods

- We use strings a lot in Python. Python provides many special built-in functions, called **methods**, for strings.
- Methods are called/invoked using a different syntax, dot notation (some people find it confusing):

```
>>> 'AbCd'.lower()  
'abcd'
```

invokes the built-in string
lower function

*NOTE: You can think of it as
lower('AbCd')*

- We will discuss string methods more next week but the lower() method is helpful for HW2

Intro to Ch8: Strings and for

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 talk about 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

Intro to Ch8: Strings and **for**

Python provides a more concise way to do the same thing as the while-loop-with-index-incrementing of the previous slide

```
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! We'll discuss more on Fri. **Note: you may but you don't *need* to use **for** for HW2. while and [] are sufficient.**

Designing while loops

When designing loops:

- Typically, just before while statement, set variables that ensure truth of boolean condition (also called “loop guard”)
- Within loop, update one or more variables of the loop guard expression. In many simple loops, this is often done at the end of the loop body.
- Carefully reason about your loop body and loop guard, convincing yourself that eventually the loop guard will become false (Note: sometimes, people formally prove these things. E.g. when validating software for, say, flight control systems)

lec7while.py while examples

- sometimes loop body never executes
- sometimes loop never terminates
 - usually this is an error, but sometimes it is on purpose
- ... and sometimes we're not actually sure whether a loop will terminate or not!

lec7while.py examples: 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() !

lec7while.py

Next time

- finish printFirstNPrimes by replacing stub isNumPrime with correct code. Develop isNumPrime in top-down fashion like printFirstNPrimes:

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

- Strings and **for**. Read Chapter 8!