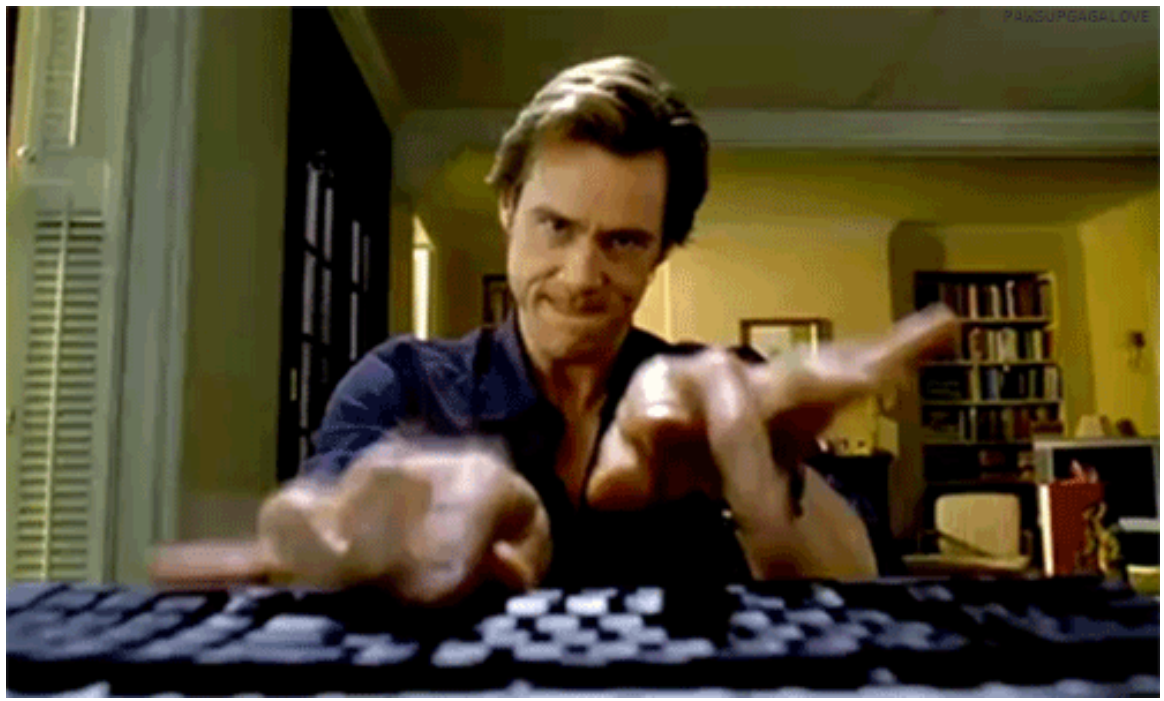
15-112
Fundamentals of Programming

Week 1 - Lecture 4:
Loops Exercises



May 19, 2016

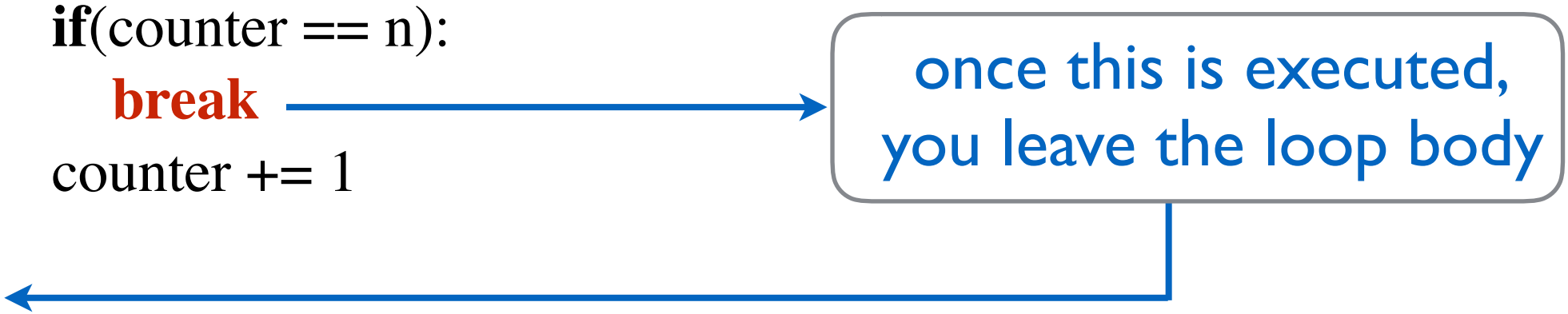
break

continue

break

Break out of the loop

```
def countToN(n):
    counter = 1
    while(True):
        print(counter)
        if(counter == n):
            break
        counter += 1
```



break

In a while loop, condition is checked at the beginning

```
while(expression):
    …
```

=

```
while(True):
    if(not expression):
        break
    …
```

Using a break statement, can check condition anywhere

```
while(True):
    …
    if(not expression):
        break
    …
```

break

Break out of the loop

```
def sumGivenNumbers():
    total = 0
    while(True):
        x = input("Enter number (or 'done' to quit): ")
        if(x == "done"):
            break
        else:
            total += int(x)
    return total

print(sumGivenNumbers())
```

continue

Break out of the current iteration

```
def sumOfOddsToN(n):
    total = 0
    for x in range(1, n+1):
        if(x % 2 == 0):
            continue
        total += x
    return total
```

once this is executed,
you skip to the next iteration

continue

Break out of the current iteration

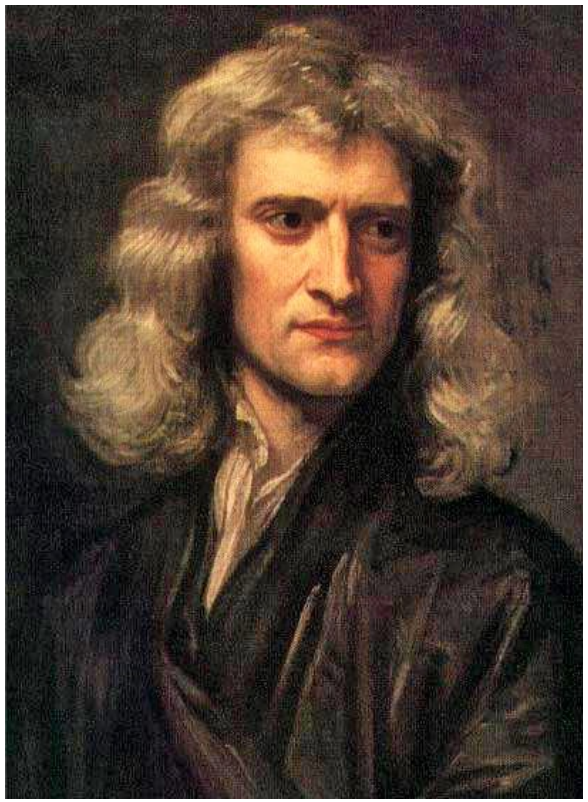
```
def multiplyGivenNumbers():
    # if 0 is given as input, we ignore it
    product = 1
    while(True):
        x = input("Enter number (or 'done' to quit): ")
        if(x == "done"):
            break
        elif(int(x) == 0):
            continue
        product *= int(x)
    return product

print(multiplyGivenNumbers())
```

Practice Problems

Computing square root

How does a computer compute square roots?

It is not a trivial thing to do!



Newton's method:

- want to compute sqrt(x)
- if y is your guess for the answer, then 0.5*(y + x/y) is a better guess.

Approximating π(n)

Prime counting function:

$$\pi(n) = \#\text{ primes} \leq n$$

Harmonic numbers:

$$H(n) = 1 + 1/2 + 1/3 + \ldots + 1/n$$

Claim: $\pi(n) \approx n / (H(n) - 1.5)$

longestDigitRun(n)

longestDigitRun(2775199923772) should return 9

longestDigitRun(2775199923777) should return 7

How do we solve this problem?

Let's slowly build up to it…

walkThroughDigits(n)

```
def walkThroughDigits(n):
    while (n > 0):
        print(n % 10)      # print rightmost digit
        n = n // 10        # get rid of rightmost digit
```

What if n == 0 ?

What if n < 0 ?

walkThroughDigits(n)

```
def walkThroughDigits(n):
    if (n == 0): print(0)
    n = abs(n)
    while (n > 0):
        print(n % 10)      # print rightmost digit
        n = n // 10        # get rid of rightmost digit
```

digitCount(n)

```
def digitCount(n):
    if (n == 0): return 1
    n = abs(n)

    while (n > 0):

        n = n // 10
```

digitCount(n)

```
def digitCount(n):
    if (n == 0): return 1
    n = abs(n)
    count = 0
    while (n > 0):
        count += 1
        n = n // 10
    return count
```

largestDigit(n)

Keep track of largest seen "so far".
Update it when you see a larger digit.

```
def largestDigit(n):
    if (n == 0): return 0
    n = abs(n)
    currentLargest = ?
    while (n > 0):
        if (n%10 > currentLargest):
            currentLargest = n%10
        n = n//10
    return currentLargest
```

largestDigit(n)

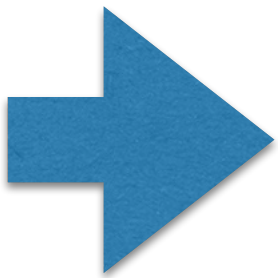
Keep track of largest seen "so far".
Update it when you see a larger digit.

```
def largestDigit(n):
    if (n == 0): return 0
    n = abs(n)
    currentLargest = -1
    while (n > 0):
        if (n%10 > currentLargest):
            currentLargest = n%10
        n = n//10
    return currentLargest
```

countOfMostFreqDigit(n)

How would you solve it?

Count the number of 0's.
Count the number of 1's.
Count the number of 2's.
…
Count the number of 9's.



Return largest count.

Suppose you have a function

digitCount(n, d):

returns the # times
digit d appears in n

countOfMostFreqDigit(n)

Algorithm:

- Loop through all the digits d = 0, 1, 2, …, 9:
- Call digitCount(n, d) to determine the count of d
- Return the largest one

```
def countOfMostFreqDigit(n):
    if(n == 0): return 1
    n = abs(n)
    currentMaxCount = 0
    for d in range(10):
        if(digitCount(n, d) > currentMaxCount):
            currentMaxCount = digitCount(n, d)
    return currentMaxCount
```

countOfMostFreqDigit(n)

Algorithm:

- Loop through all the digits d = 0, 1, 2, …, 9:
- Call digitCount(n, d) to determine the count of d
- Return the largest one

```
def countOfMostFreqDigit(n):
    if(n == 0): return 1
    n = abs(n)
    currentMaxCount = 0
    for d in range(10):
        dCount = digitCount(n, d)
        if(dCount > currentMaxCount):
            currentMaxCount = dCount
    return currentMaxCount
```

mostFreqDigit(n)

```
def mostFrequentDigit(n):
    if(n == 0): return 0
    n = abs(n)
    currentMaxCount = 0
    currentMostFreqDigit = -1
    for d in range(10):
        dCount = digitCount(n, d)
        if(dCount > currentMaxCount):
            currentMaxCount = dCount
            currentMostFreqDigit = d
    return  currentMostFreqDigit
```

isSubnumber(n, m)

Is m a subnumber of n?

m = 1234

n = 123983257214570495731038450398071102
98545793847503978503984750928746 1248
91327490128751287398572093847502938 4
75091860180801964816439817634013041 7
65981726581641028364107561287365 1982
37461028576 1982576 1247460 1239764 1285
67102467109237480912352847569385 7632
98576340583092458720934875018347 5945

isSubnumber(n, m)

Exercise: Write the code

longestDigitRun(n)

longestDigitRun(2775199923772) should return 9

longestDigitRun(2775199923777) should return 7

n = 123983257214570495731118450398071102
98545793847503978503984750928746 1248
91327490128751287398572093847502938 4
75091860188801964816439817634016666 6
66981726581641028364107561287365198 2
37461028576198257612474601239764128 5
67102467109237480912352847569385763 2
98576340583092458755555500777775555

longestDigitRun(n)

Suppose you had a function

digitRunAtStart(n) that returns length of the "initial" run.

n = 123983257214570495731118450398071102
98545793847503978503984750928746 1248
91327490128751287398572093847502938 4
75091860188801964816439817634016666 6
66981726581641028364107561287365198 2
37461028576198257612474601239764128 5
67102467109237480912352847569385763 2
98576340583092458755555500077777 5555

4

longestDigitRun(n)

Suppose you had a function

digitRunAtStart(n) that returns length of the "initial" run.

n = 123983257214570495731118450398071102
98545793847503978503984750928746 1248
91327490128751287398572093847502938 4
75091860188801964816439817634016666 6
66981726581641028364107561287365198 2
37461028576198257612474601239764128 5
67102467109237480912352847569385763 2
98576340583092458755555000 77777

5

longestDigitRun(n)

Suppose you had a function

digitRunAtStart(n) that returns length of the "initial" run.

n = 123983257214570495731118450398071102
98545793847503978503984750928746 1248
91327490128751287398572093847502938 4
75091860188801964816439817634016666 6
66981726581641028364107561287365198 2
37461028576198257612474601239764128 5
67102467109237480912352847569385763 2
98576340583092458755555 000

3

longestDigitRun(n)

Suppose you had a function

digitRunAtStart(n) that returns length of the "initial" run.

```
def longestDigitRun(n):
    if(n == 0): return 0
    n = abs(n)
    currentLongestRun = 0
    currentLongestDigit = -1
    while(n > 0):
        digit = n %10
        if(digitRunAtStart(n) > currentLongestRun):
            currentLongestRun = digitRunAtStart(n)
            currentLongestDigit = digit
        n = n // 10**digitRunAtStart(n)
    return  currentLongestDigit
```

longestDigitRun(n)

If there is a tie, smaller digit should be returned.

Exercise: fix the code to accomplish this.

isRotation(x, y)

Input: non-negative integers x and y

Output: True if x is a rotation of the digits of y,
False otherwise.

e.g. 3412 is a rotation of 1234

321 is a rotation of 3210

Algorithm:

- Let k = # digits of y
- Repeat k times:
  - Rotate y by 1 digit.
  - If x = y, then return True
- Return False

isRotation(x, y)

- Let k = # digits of y
- Repeat k times:
  - Rotate y by 1 digit.
  - If x = y, then return True
- Return False

```
def isRotation(x, y):
    for i in range(digitCount(y)):
        y = rotateOnce(y)
        if(x == y): return True
    return False

def rotateOnce(n):
    return (n%10) * 10**(digitCount(n)-1) + n // 10
```

isRotation(x, y)

There is a bug!

Exercise: Find and fix.

nthPalindromicPrime(n)

A number is palindromic if it reads the same backward or forward.

e.g. 24342, 989, 5

```
def nthPalindromicPrime(n):
    found = 0
    guess = 0
    while (found <= n):
        guess += 1
        if (isPalindromicPrime(guess)):
            found += 1
    return guess
```

nthPalindromicPrime(n)

A number is palindromic if it reads the same backward or forward.

e.g. 24342, 989, 5

```
def nthPalindromicPrime(n):
    found = 0
    guess = 0
    while (found <= n):
        guess += 1
        if (isPalindromic(guess) and isPrime(guess)):
            found += 1
    return guess
```

nthPalindromicPrime(n)

```
def isPalindromic(n):
    return (n == reverse(n))



def reverse(n):
    nReversed = 0
    while (n > 0):
        # construct nReversed
        n = n//10
    return nReversed
```

nthPalindromicPrime(n)

```
def isPalindromic(n):
    return (n == reverse(n))



def reverse(n):
    nReversed = 0
    while (n > 0):
        nReversed = nReversed*10 + n%10
        n = n//10
    return nReversed
```