# Exam 1 Review

15-110 – Monday 02/17

# Announcements

- Check3 was due today
- Check2/Hw2 revision deadline **tomorrow (Tuesday) at noon!**
- No Gradescope exercise today (no new material)
- **Exam1 on Wednesday!**
  - Bring your paper notes (<= 5 pages), something to write with, and your andrewID card
  - Arrive **early if possible** – we're checking IDs at the door

# Announcements – Code Reviews

- **Code reviews!**
  - **What:** meet with a TA for 10-15 minutes to get qualitative feedback on your code from your Hw2 submission. Attending the meeting and actively participating gets you 5 points on Hw3.
  - **Why:** code style and structure are important, but not assessed by the autograder. The TA will point out different ways to solve the problems and areas where you can code more clearly or more robustly
    - Some students may be exempted from this meeting if they already have good style. We'll let you know if you're in that group by Monday EOD.
  - **When:** this weekend (Saturday-Sunday, a few slots on Monday)
  - **Where:** TA's choice

- How to sign up for a code review slot
  - **Link:** https://www.cs.cmu.edu/~110/hw/hw2-code-review.html
  - **Important:** sign-ups for each TA slot close 5pm Friday
  - **Also important:** don't be late! If you are more than 3 minutes late to your meeting, you will not get credit on Hw3.
    - If something comes up and you need to cancel, notify the TA at least an hour before your timeslot. Do not do this multiple times.

# Review Topics

- For Loops
- While Loops
- Nesting

# For Loops

# For Loops

A **loop** is a control structure that lets you repeat a number of statements (the **body** of the loop) a certain number of times.

A **for-range loop** implements this looping by setting the loop control variable to a **pre-determined set of numbers**. The numbers are generated by the **range** expression.

We usually use for loops when we know **exactly how many times we need to loop**.

# For Loop Structure

Recall that the basic structure of a for loop is:

```
for variableName in range(start, end, step):
    LoopBody
```

Here's a simple example:

```
for num in range(1, 10): # step defaults to 1
    print(num)
```

# Example: Code Reading

Consider the following code snippet:

```
count = 0
for x in range(1, 101):
    if isPrime(x):
        print(x)
        count = count + 1
print("Total:", count)
```

If we assume that `isPrime` has been written and works correctly, what does this do?

# For Loops over Strings

A common pattern is to loop over each character of a string and check a property or use the character in a computation.

Here's an example:

```
def allA(s):
    result = ""
    for i in range(len(s)):
        if s[i] == "a":
            result = result + s[i]
    return result
```

What will this function return if called on the string `"banana"`?

# Activity: `findMatches`

**You do**: Write `findMatches(s1, s2)`, which takes two same-length strings and returns `True` if they ever have the same character at the same index, and `False` otherwise.

Examples:

`findMatches("apple", "guava")` returns `False`.

`findMatches("apple", "grape")` returns `True`, because the `e`s match.

# While Loops

# While Loops

For-range loops are useful for a majority of cases, but there are some situations where we can't set up a loop control variable in a straightforward way.

**While loops** are useful for these more complex cases. They allow us to manage the loop control variable **directly** instead of having the for loop manage it for us.

# While Loop Structure

Recall that the basic structure of a while loop is:

```
while condition:
    loopBody
```

This is much simpler than a for loop, but also means we'll need to set up a variable before the loop and update it inside the loop. For example:

```
num = 1
while num < 10:
    print(num)
    num = num + 1
```

# Example: Quizlet3

Complete the function below named `printPowersOf2` that takes a parameter `x` and prints the powers of 2 up to `x` inclusive. For example:

`printPowersOf2(7)` prints:

```
>>> printPowersOf2(7)
1
2
4
```

and `printPowersOf2(16)` prints:

```
>>> printPowersOf2(16)
1
2
4
8
16
```

```
def printPowersOf2(x):
    power = _____________
    while _____________:
        _____________________
        _____________________
```

# Example: Code Writing

Let's write a function that displays all the Fibonacci numbers up to and including a given number n.

```
def fibUpToN(n):
    prior1 = 0
    prior2 = 1
    print(prior1)
    print(prior2)
    current = prior1 + prior2
    while current <= n:
        print(current)
        prior1 = prior2
        prior2 = current
        current = prior1 + prior2
```

Note that **current** is our loop control variable. It starts as the first Fib number we can make out of two others, is set to the next Fib number in each loop, and we check if it's gotten too big in each condition.

# Activity: Mystery Code

**You do:** trace the given code. What does it do?

**Hint:** consider what happens when we call `mystery("lets go tartans", "t")`

```
def mystery(s, c):
    i = s.index(c)
    while i < len(s):
        s = s[:i] + s[i+1:]
        if c in s:
            i = s.index(c)
        else:
            i = len(s)
    return s
```

# Nesting

# Nesting Changes a Program's Control Flow

**Nesting** is the process of indenting control structures so that they occur inside other control structures. It is used to manipulate the control flow of a program to produce certain intended effects.

So far, we've learned about several control structures: **function definitions, conditionals, while loops,** and **for loops**. All of these structures have **bodies**, and each can be indented so it occurs inside the body of another structure.

# Common Nested Structures - Functions

Though any nesting configuration you can think of is possible, some arrangements are more common than others.

**Functions** – we usually write function definitions at the top level of a program and nest conditionals/loops inside them when they're needed. When we **return** in a nested conditional/loop, we exit that structure and the whole function immediately.

```
def hasVowels(s):
    for i in range(len(s)):
        if s[i] in "aeiou":
            return True
    return False
```

Note how the loop is indented inside the function, and its body is indented again.

If the line `return True` is reached, the function will exit immediately without finishing the loop.

# Common Nested Structures - Functions

It's also common to include a function call inside the definition of another function.

**You do**: what will this print?

```
def foo(a, b):
    y = a + b
    print("y in foo:", y)
    return y + 3

def bar(x):
    y = x + 1
    print("y in bar:", y)
    return foo(x, y)


print(bar(4))
```

# Common Nested Structures – Loop-Conditionals

**Loop-Conditional** – very often we nest a conditional inside a loop to check a certain property for every element that is iterated over.

While it's possible to pair an else with the nested if, it's only used if there's a clear alternative action. It's okay to do nothing on iterations that don't meet the requirement!

```
def countVowels(s):
    result = 0
    for i in range(len(s)):
        if s[i] in "aeiou":
            result = result + 1
    return result
```

We don't need to update result if the letter isn't a vowel, so do nothing instead.

# Common Nested Structures – Nested Loop

**Nested Loop** – if you need to iterate over multiple dimensions, a nested loop (one loop nested inside another) will manage the complex iteration. Each loop control variable manages one dimension.

It's important that the two loop control variables have different names, so that they can be referred to separately.

The outer loop moves more 'slowly', as it only iterates once for each complete working of the inner loop.

```
def coordinates(x, y):
    for xNum in range(x):
        for yNum in range(y):
            print("(" + str(xNum) + ", " +
                        str(yNum) + ")")
```