Exam 1 Review

15-110 – Monday 02/20

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. Prof. Kelly will let you know if you're in that group before sign-ups are released.
  - **When:** this weekend (Saturday-Sunday)
  - **Where:** TA's choice

- How to sign up for a code review slot
  - **Link:** TBA on Piazza
  - **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

- Strings
- For Loops
- Error Messages

Strings

Indexing

In a string (or list), each character (item) has a specific **position**. Positions start at `0` and go to `len(value) - 1`.

You can access an individual character from a string (item from a list) with **indexing,** using square brackets with something that evaluates to an integer.

```
s = "studying"
s[3] # "d"
x = 5
s[x] # "i"
```

Slicing

You can also extract a substring from a string (or sublist from a list) using **slicing**. Slicing uses square brackets with colons in between to specify the **start, end,** and **step** of a slice.

If any of these three components are left blank, they evaluate to a default value – `0` for start, `len(value)` for end, `1` for step. If the step is left to a default, the second colon can also be removed.

```
lst = ["Ready", "for", "the", "exam"]
lst[0:len(lst):2] # [ "Ready", "the" ]
lst[2:] # [ "the", "exam"]
```

You Do: Practice Coding with Indexing/Slicing

You do: write several short lines of code using indexing and slicing to solve problems.

1. What slice would remove the first and last characters from a string `s`?
2. What index would access the middle character from a string `s`? (You can assume it has odd length).
3. What expression would produce a 'doubled' version of a string `s`; for example, the string `"coding"` would become `"codcodinging"`?

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**.

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?

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!

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

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

Example: Code Writing

Now let's write code with a for loop. Let's write the function `countDoubleLetters` which counts the number of double-letters that occur within a string.

For example, `countDoubleLetters("bookkeeper")` would return `3`, because there are three sets of double letters (`"oo"`, `"kk"`, and `"ee"`).

Error Messages

Three Error Types

Syntax Errors – Python can't parse the code you've written into a meaningful structure

- Illegal tokens, jumbled token order, missing tokens

Runtime Errors – Python can parse the code you've written, but something goes wrong when it tries to run the code

- Type mismatch, out-of-bounds error, unknown variable

Logical Errors – Python can parse and run the code you've written, but the result isn't what you wanted

- Caught using test cases with `assert`

Practice with a Quiz!

Let's do a quick quiz to test your knowledge of error types.

For each piece of code, vote for whether it will cause a Syntax Error, Runtime Error, Logical Error, or no error.

#1

$$x = \#4$$

#2

$$y = (2) + (7)$$

#3

```
z = 5
print("Variable x =", z)
```

#4

```
x = "cake" / 2
```

#5

```
x = 2 plus 2
```

#6

```
print( str( int(42.5) ) )
```

#7

```
s = "ABC"
c = s[10]
```