6.0001 Recitation 1 - Spring 2019

Feb 7

I. Administrivia

Recitation Times

- 10 AM, 11 AM, and 1 PM (5-134)
    - Will review lecture material from that week
    - Notes will be posted on Stellar after recitation on Fridays

Course Website

- Link: https://sicp-s1.mit.edu/spring20/
- Used for course materials, calendar, psets (submitting and grading), help queue, checkoffs, everything really.
    - Grades from the pset autograder will be released 3 days after the deadline

Ed Forum

- Link: https://us.edstem.org/courses/272/discussion/
    - There are separate forums for 6.0001 and 6.0002 halves
- Q&A forum, best way to get a fast response
- If you have a specific question (makeup, psets grades, etc) make a private post
    - Please use this as opposed to emailing the staff email (emails can get lost, etc.)

Stellar

- Used for course announcements

Office Hours (38-370)

- Monday - Thursday (11am - 9pm, except during lecture time); Friday (11am - 5pm)
- Come in to get help on psets, lecture material, and pset checkoffs
- Queue: https://sicp-s1.mit.edu/spring20/queue
    - (need certificates, talk to a TA in office hours if you're having trouble logging in)

Problem Sets

- Collaboration: don't plagiarize. write your own code
- Pset 1 is out, due Wednesday 2/12 at 5PM
- Check-offs start with Pset 1, **no check off for Pset 0**
- **Cannot use late days** on PS0
- The last submitted pset is used for grading & late day calculation
- Submit on course website - (Need to be logged in) https://sicp-s1.mit.edu/

Late Days

- 3 late days in total

- Can use up to 3 per pset
    - 1 late day = 24-hour extension
    - Late days are discrete (no half late day/12-hour extensions)

MITx

- Link on Stellar
- Has **mandatory** finger exercises, which will help you solidify important concepts with small, relatively quick problems
    - Graded for accuracy
    - Lots of them - one poor score won't kill you! :)
    - Due before each lecture
    - Also contains optional exercises for extra practice

Checkoffs

- Starting with PS1, you need a checkoff for each pset (generally worth 30% of your overall pset grade)
- Usually due 10 days following the posted due date of the problem set. Check pset doc or calendar for specific due dates.
- **Late days cannot be used for a late checkoff**
- You will go through your code with a TA or LA. They will ask you simple questions about your code and determine a score based on code style and understanding of the pset and code.
- Generally speaking the closer you are to the checkoff deadline, the longer the queues in OH, so get them done early!!

How to Succeed

- Sign up for HKN Tutoring
- Read the Style Guide! You want to make sure your code is easily understood by others
    - In the real world, there are lots of rules for how code should be formatted
    - Paying attention to how your code looks is just as important as functionality
    - Many deductions on psets are because students don't read the style guide (available on Stellar)
- Practice Practice Practice!!
    - Reference links on Stellar for more Python practice (Python Resources under Materials)

II. Intro to Python and Anaconda/Spyder

Anaconda

- Anaconda is a *Python Distribution*, which in one installable package contains Python, a set of Python packages, a code editor (Spyder), and an interactive interpreter/shell (IPython)

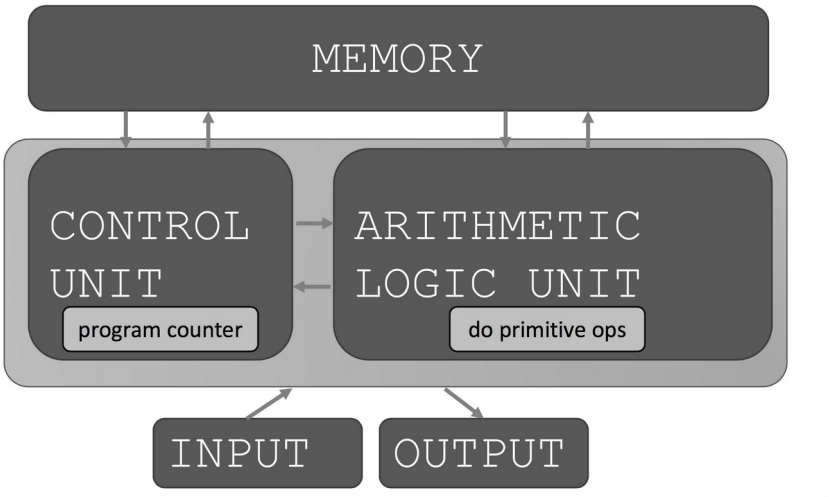
Spyder

- Spyder = Scientific PYthon Development EnviRonment (a place to edit code, run it, and debug it)
- This is the development environment encouraged by this class, though there are others
- IPython shell vs new window/opening a file:
    - The shell is interactive and will give you results right away at each step as you type it
        - use mostly for testing things out - try it before you ask about it!
        - has a `help` command (like `help(str)`)
        - can have multiple shells open at once via Consoles > Open an IPython console
        - If your console disappears, go to Consoles > Open to get it back!
    - use File > New File... to create a new file to run later and get the results all at once after you're done writing it
        - can do more complex code this way than in the shell
        - can also open .py files you already have and run them
        - These results are only printed out if you explicitly use the `print` command, unlike in the shell where the result is printed either way
- Saving your File
    - Hit File > SaveAs to name your file.
    - When you hit the green Run button, your work at that point is saved!
- **Make sure you run your code before you turn it in.** Anaconda/Spyder saves all variables - if you use x = 5, and then delete x later, Spyder will still know what x is
    - make sure you open a new console and try to run your code before you turn it in!
    - You can also restart your kernel - it will have the same effect
    - There are instructions in the Getting Started PDF from PS0 on how to prevent this from happening

<u>What does a Computer do?</u>

- Performs built-in and defined calculations and remembers results
- Computes can only do what you tell them to do

BASIC MACHINE ARCHITECTURE



Keep in Mind:

- Computers are dumb. They can only calculate and remember things. They are very good at this
- Your computer will do EXACTLY what you tell it to - no more, no less.

Python

- general-purpose, high-level language that is widely-used
- a program in Python (and other languages) is a sequence of expressions, or instructions
- expressions are sequences of operands and operators
- Whitespace matters in Python - be careful with indentation and use the tab key, not the spacebar (when you have long, nested code, it's hard to tell the difference between 3 and 4 spaces, 7 and 8 spaces, 11 and 12 spaces, etc)
- use the Python documentation: https://docs.python.org/3.6/library/index.html

Variables

- Naming convention: snake_case
- Must start with a letter or underscore (_)
- Can only contain alphanumerics & underscores
- Can't use reserved keywords: https://docs.python.org/3/reference/lexical_analysis.html#keywords
- order of evaluation is right side then left side ; ex:

```
>>> a = 5
>>> a = a + 5
# prints that a is 10
```

Type

- everything in python is an object, and objects have types
- Basic (“primitive”) types: int, float, string, boolean
- other types to know: NoneType (None)
- Built-in function type(some_object) will tell you its type
    - Other built-in functions: https://docs.python.org/3/library/functions.html
- be careful of type issues
    - 1 / 2 = 0.5 (float division)
    - 3 // 2 = 1 (integer/floor division)
    - 1 / 2.0 = 0.5 (float division)
    - 3. // 2 = 1.0 (integer division, cast to float)
    - float(1) (casting)

Operations

- Arithmetic operations follow PEMDAS rules
    - +, -, *, /
    - ** for exponents (note: ^ is bitwise XOR - be careful!)
    - % “modulo” or “mod” to get remainder
- String operations (overloading arithmetic operators)
    - + for concatenation
    - * to repeat

Loops

- `for` loops have pre specified range over which they run.
    `for i in range(x):`
        - `i` goes from 0 to x-1

    `for char in s:`
        - `char` is string that takes on the value of each character in `s`
- `while` loops have a condition that they check to determine if they should keep running. They run until the condition no longer evaluates to True.

```
counter = 0
while counter < 3:
    print(counter)
    counter += 1
```

- Converting between for and while loops
    - All for loops can be written as while loops
    - Not all while loops can be written as for loops

Output

- `print(x)`
- `print (“x = ”, x)` (comma concatenates with a space between)
- print statements are super useful for debugging! especially to see what is happening in loops
    - print out intermediate variables/values to trace what is going on
- Ex: Will this work? If it does, what does it print?
    - `print(“hello” + “world”)`
        - `“helloworld”`

    - `print("hello" , "world")`
        - `"hello world"`
    - `print("hello" , "wor" + "ld")`
        - `"hello world"`

Input

- `x = input("user prompt ")` → x is a string
- Remember to save user input to a variable if you want to use it later
- Remember to cast it to the type that you want
    - `integer = int(input("enter an int"))`

III. Style

- Choose descriptive variable names
    - You will lose points on your psets if you don't!
- Comments
    - `# single line`
    - `""" multiple line docstring for documentation purposes """`
        - Can use single ‘’’ ‘’’ or double “”” “”” quotes (3 on each side of same kind)
    - In Spyder, Edit > Comment/Uncomment
    - Be descriptive but concise - don't need paragraphs or comments on every line
    - Comments look best when put **before** the line or block of code they refer to, and with the same indentation
- Why?
    - Helps explain your code
    - Easier to debug & understand - especially if someone else is reading it
    - Real Life coders have rules - there are conventions that you are expected to follow when working at companies!
    - You will lose points on your psets if you don't :)
- **Read style guide to avoid deductions on psets**