Computer Science — Python — HW #13

Assigned on Day, YYYY-MM-DD. Due on Day, YYYY-MM-DD.

1. Read chapter 17 of ThinkPython, 2nd edition. (Only one chapter this week, so you can read it twice!)
2. Read the handout called Brief Review of Classes. Make sure that you understand everything in it. In fact, make sure you can write out both the classes presented there (Player and TeamPlayer) without referring to notes.
3. Decide on a course project from the nine ideas that you've up with. Or decide on a new, tenth idea. Think about the details of the project, including the top three things (i.e., "features") you'd want in the project, and the three aspects of the project that you're the least certain about (e.g., details you're not sure you'll get to, or things that could go wrong). A homework assignment **next week** will involve formulating a Project Plan.

Remember that you're welcome to collaborate with others by forming groups/teams/posses, as long as the Project Plan makes clear who is responsible for doing what. Note that collaboration can take many forms….

1. Suppose two teams each want to write an adventure-type program. They could decide that there are some classes that they could share. One team could write half of the shared classes, while the other one could write the other half of the shared classes.
2. Here's a variation on (a). Maybe a third team could write all the shared classes, allowing the two original teams to use those classes, and devote more time to their own individual adventures.
3. **[Turn in]** Trace the execution of the following scripts by completing the table below. Write down what happens at each line of execution. The Line #s in the table represent the line numbers of the Python script.

(It might be easier to write this up in a Microsoft Word document, or in some other format that supports tables.)

(a) **Banana slices** (Based on problem #5 of HW #7).

**Note:** The script to the left has 4 lines, so all the Line #s in the table in part (a) should lie between 1 and 4.

Line #1: s = 'banana'

Line #2: for k in range(0, len(s)):

Line #3: print(s[0])

Line #4: s = s[1::]

|  |  |  |  |
| --- | --- | --- | --- |
| **Line #** | **Change in memory** | **Script output** | **Notes** |
| Line #1 | s = 'banana' | No output |  |
| Line #2 | k = 0 | No output | Starting loop |
| Line #3 | No change | Output: b |  |
| Line #4 | s = 'anana' | No output |  |
| Line #2 | k = 1 | No output | Back to top of loop |
| Line #3 | No change | Output: a |  |
| Line #4 | s = 'nana' | No output |  |
| ...etc... (fill out the rest of this table) | | | |

**(b) (E-)Counting your feet** (Based on problem 2(b) of Quiz #1).

Line #1: def ecount(s):

Line #2: count = 0

**Note:** All the line numbers table in part (b) should lie between 1 and 7.

Line #3: for character in s:

Line #4: if character == 'e':

Line #5: count += 1

Line #6: return count

Line #7: print(ecount('feet'))

|  |  |  |  |
| --- | --- | --- | --- |
| **Line #** | **Change in memory** | **Script output** | **Notes** |
| Line #1 | Function ecount is defined | No output |  |
| Line #7 | ecount is called with s='feet' | No output | Entering function |
| Line #2 | count = 0 | No output |  |
| Line #3 | character = 'f' | No output |  |
| Line #4 | No change | No output | Condition is False |
| Line #3 | character = 'e' | No output |  |
| Line #4 | No change | No output | Condition is True |
| ...etc... (fill out the rest of this table) | | | |

NWYH — Computer Science — Python — HW #14

Assigned on Day, YYYY-MM-DD. Complete by Day, YYYY-MM-DD.

1. Read chapters 12 & 13 of ThinkPython, 2nd ed.
2. **[Turn in]** Write up a draft of a **Project Plan** for your course project. It should contain the following sections:

(A Microsoft Word doc is recommended, or a comparable format that supports tables and comments.)

This should have enough detail to convey what you have in mind, without getting bogged down in details.

1. **A**bout the Author, Activity, and Audience
   * Project name (e.g., DailyDo for a TODO-list, or Communitas for a political awareness app)
   * Who (One person, or multiple, with a group/team/posse name). Also, who's the app for?
   * What. Brief description or elevator pitch. If you're on a team, who will do what?
2. **B**eginnings
   * What do you need for success that you might not already have? For example: a working laptop, tkinter or similar app examples, trig/physics understanding, etc. Let's see you get it.
3. **C**ore features and stretch goals (at least 3). (If you're on a team, who will work on what?)
4. **D**esign. What components the system has, what classes you'll use, and how they're tied together.
5. **E**volution. What features you hope to accomplish, by week (see table, below).

(Don't panic — You won't be held to this schedule, but you will be asked to manage it. If you drift from the schedule, you can use the amount of drift to adjust it. For example, if you're only going at half the pace that you originally anticipated, then you can cut out half the planned functionality, or make other compromises or trade-offs to improve productivity.)

|  |  |  |
| --- | --- | --- |
| Week (Mon-Thurs) | Calendar Notes | Project goals — **TODO: Fill in this column.**  Status updates are due each Thursday. |
| 2017 week #17  (Apr 24 – Apr 27) | Quiz #3 on Wed, 2017-04-26 | (HW #17 = Provide status update)  E.g.: Which features done? Which classes written? |
| 2017 week #18  (May 1 – May 4) | **Note:** Mon is Yom Hazikaron; Class trip starts end of week | (HW #18 = Provide status update)  ..Did any risks materialize? |
| 2017 week #19  (May 8 – May 11) | **Note:** Returning from trip | (HW #19 = Provide status update)  ….Work through any nasty bugs? |
| 2017 week #20  (May 15 – May 18) |  | (HW #20 = Provide status update)  ……Did you decide to change your project? |
| 2017 week #21  (May 22 – May 25) | Should aim for completing project this week | (HW #21 = Provide status update)  ……..Completed everything you set out to? |
| 2017 week #22  (May 29 – June 1) | **Note:** Mon is Memorial Day; Wed is Shavuot | HW #22? |
| 2017 week #23  (June 5 – June 8) | **Note:** Mon is last class day;  Wed is Merit Awards | HW #23? |

1. **F**ooey! Biggest risks (three or more): What could go wrong & ruin the project. (**TODO:** Fill in table.)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Risk name | Description | Likelihood | Impact | Response (If the risk does come to pass, how can you respond to save the day?) |
| Slow turtle | turtle might not support video-game-like speeds | Low/Med/High  L= < 10%  M= 10-25%  H= > 25% | L/M/H  L= < 1d  M= ~1d  H= > 1d | Use tkinter instead of turtle graphics. |
| Multiple files | Might need to use multiple Python files | H | L | Find out how to run Python code w/ many files. |
| … | … | … | … |  |

1. **G**lossary. Define any terms you're using that readers of this document might not already know.

NWYH — Computer Science — Python — HW #15

Assigned on Day, YYYY-MM-DD. Complete by Day, YYYY-MM-DD.

1. Reach chapters 14, 18, and 19 of ThinkPython, 2nd edition. (Spoiler alert: Next homework covers the Appendices.)
2. **[Turn in]** **Course project review.** Pair up with another person or team, and review each other's Project Plan. Everyone must review at least one Project Plan from another person or team. Provide at least three suggestions for change to the other team, and copy me on the communication.
3. **[Turn in]** **Course Project Proof of Concept.** The hardest step is the first step. Submit a short program that is a first programming step for your project.
   * If your Course Project is to use turtle graphics to draw between 50 and 100 national flags, then a good first step could be to write a program that draws a single national flag.
   * If your Course Project is an app that displays Hebrew text using tkinter, then a good first step could be a program that displays the text "!שלום, עולם" in a window when you click the left button labelled "Hello".
   * If your Course Project is a text-based Tic-Tac-Toe game that the user plays against the computer, then a good first step could be a program that prints out a board with a few pieces on it.
   * If your Course Project is a text-based Monopoly game, then a good first step could be to write a game loop that alternates the players taking turns. Each player could roll a pair of dice, and the values could then be ignored. Why would they be ignored? Because it's just a first step---not the final program.
4. **[Turn in]**
   1. Write a Python script that uses Python's turtle module to draw a stick figure, with at least a head, torso, 2 arms, and 2 legs. The stick figure can be in the traditional "stick figure" pose, as shown below to the left, or it can be shown walking, running, leaping, or rehydrating with iced caffè latte after a grueling game of hockey or lacrosse. Feel free to make it distinctive by adding bling, swag, or extra features, such as hands, feet, shoulders, a hat, etc.
   2. Write a Python script that uses Python's turtle module to draw a face, with at least something roundish to represent the head, as well as two eyes and a mouth. The nose is optional. It can be a smiley face or a "Have a Day" face, as shown below and to the right. Feel free to make it distinctive by adding features, such as hair, ears, lips, teeth, eyelashes, a furrowed brow, freckles, dimples, a beard, peiyot, etc.



1. **[Optional] Lunar Lander.** Here is a game for you. Download **lunar\_lander.py.zip** from the course website, and run it a few times until you're reasonably good at it. Can you start at 300 feet, and land safely, using only 100 fuel units?

* Lunar lander is an example of a script that handles command-line arguments. You can call the script in a way that sets the initial height, velocity, or amount of fuel remaining. And you can peek inside the script to see how you can handle command-line arguments in your own program.

python lunar\_lander.py --help # This (w/ two hyphens) causes the game to print out a help message.

python lunar\_lander.py --height 300 --velocity 20 --fuel 100 # Set initial parameters (w/ double hyphens)

1. **[Optional; Warning: Difficult] Spoken digits.** I'm sure you can imagine a program that listens to spoken input, and converts spoken numbers into integers. Maybe it's setting a radio station, or maybe setting the address of a destination to navigate to. Download the script **spoken\_digits.py** from the course website. This script uses an algorithm (described in the script) that attempts to perform part of this conversion. It works for one out of three test cases, but fails on the other two. Can you modify this algorithm to work for all three, or come up with an entirely new algorithm that works?