Computer Science — Python — HW #9

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

1. Read chapter 8 and 9 of Think Python, 2nd ed. (Continuing with two chapters per week.)
2. **Lambdas, map, and filter.** Recall that

**map**(f, items) behaves like [ f(x) for x in items ]

and that

**filter**(p, items) behaves like [ x for x in items if p(x) ]

Figure out the output of the following script. Then run the script to confirm your understanding.

from random import randint

values = [ 'A','1','2','3','4','5','6','7','8','9','10','J','Q','K' ]

suits = [ 'C', 'D', 'H', 'S' ]

deck = [ (value, suit) for value in values for suit in suits ]

positions = [ randint(0, 51) for k in range(0, 7) ]

cards = [ deck[pos] for pos in positions ]

to\_value = **lambda** c: c[0]

to\_suit = **lambda** c: c[1]

def print\_card(c):

print('{0:s} of {1:s}'.format(c[0], c[1]))

for c in cards:

print\_card(c)

print('=====')

for h\_card in **filter**(**lambda** c: to\_suit(c) == 'H', cards)

print\_card(h\_card)

print('=====')

for v in **map**(to\_values, cards)

print('Value: {0:s}'.format(v))

print('=====')

for s in **map**(to\_suits, cards)

print('Suit: {0:s}'.format(s))

1. **[Turn in]** Course project brainstorming. Submit descriptions of three candidate ideas for your course project. The descriptions should be "elevator pitches" — brief enough to be spoken in a brief elevator ride, but long enough to convey an idea of what the project would entail and possibly what makes it interesting. If it were to be an elevator pitch for a movie, this amount of detail would be OK:

The year is 1895.  In an inventors' laboratory co-op, just after the Frankenstein monster is brought to life, he accidentally walks into the active portal of a just-invented time machine next door.  In the distant future, he meets up with a super-intelligent talking dog named Quetzal. Opens in theaters July 5, 2019.

**Note:**

* 1. You are welcome to collaborate with others in the course, but there must be a clear separation/demarcation of responsibilities within each team: Who is going to design and write which portions of the final project.
  2. Your project doesn't need to be written from scratch. There are plenty of Python libraries and frameworks that are publicly available. For example:
     + Tkinter is a "windowing toolkit" that can be used to create a desktop app. We'll be briefly covering this in Unit #3 of the course. (See <https://docs.python.org/3/library/tk.html>)
     + Pygame is a set of Python modules for writing video games ([www.pygame.org](http://www.pygame.org)).
     + If you search online for "python chat client", you can find some sample programs that could form the basis of inter-computer communications.
  3. Don't worry too much about the scope of the project right now. Small ideas can be elaborated and enriched as needed, while overly ambitious ideas can be pared down.
  4. We'll have a couple more homework assignments in Unit #2 focused on brainstorming, so this isn't the final chance to come up with a project idea.
  5. In Unit #3 of the course, you'll create a project plan for your course project. That might involve some research, including writing one or more small proof-of-concept programs. Even then, you can change the project, though it will take more effort than settling on a project in Unit #2.

NWYH — Computer Science — Python — HW #10

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

1. Read chapters 10 and 11 of Think Python this week. (Continuing with two chapters per week.)
2. **[Turn in]** "Craps". [Please think about this before Wednesday, 2016-03-08. We'll work on this in teams in class.]

Here are the (slightly simplified) rules of each "round" of a dice game called craps.

* A player starts out by rolling a pair of six-sided dice.
* If the first roll (i.e., the sum of the two dice) is a 2, 3, or 12, then the player loses.
* If the first roll is a 7 or 11, then the player wins.
* If the first roll isn't any of those listed above, then the first roll becomes the player's "point", and play continues, with the player repeatedly rolling the same pair of six-sided dice. For this continued play:
  + If the player rolls a 7 before rolling the "point", then the player loses the round.
  + If the player rolls their "point" before rolling a 7, then the player wins the round.
  1. Download the script **craps.py.zip** from <http://www.nyhs.org/Page/661>
  2. Fill in the code for **get\_round\_outcome**, so that it plays a round of Craps, and returns either **LOSE** or **WIN**, as appropriate.
     1. **Hint:** Make a list of the information you need to track (e.g., the player's "point"), and then name variables that you will introduce.
     2. Recall how the **'in'** operator works in Python:

3 in [1,2,3] # Value is True

5 in [1,2,3] # Value is False

* 1. Submit your modified version of the script **craps.py** (as an **attachment**, of course), together with the output from running it (i.e., the line printed out at the end of the script, after the call to **play\_set**).

1. **[Turn in]** Find-the-pebble.

I wrote the first version of a game I'll call Find the Pebble. But there's a bug in the initial text version. Specifically, it seems to take way too long to find the pebble. Boo!

The idea of the game is that the user first chooses the dimensions of a game board. Then the computer chooses the location of a pebble that will stay at a fixed location on the board while the player chooses locations by specifying x and y coordinates. For each guess the user makes, the computer tells the user how far off their guess is, in terms of Cartesian distance (i.e., the square root of the sum of the squares of the x- and y- distances).

It seems impossible to win the game for a large board. If I choose a smaller dimension, I eventually win, but it still seems to take forever! There might be a problem with the distance function, but maybe not.

* 1. Download the script **find\_the\_pebble.py.zip** from <http://www.nyhs.org/Page/661>
  2. Take a look at the code and run the script. See if you agree that it doesn't run the way you'd hope it would.
  3. Fix whatever the problem/bug is, and submit the corrected version of **find\_the\_pebble.py** as an **attachment**.
  4. You're welcome to collaborate on this, but please don't reveal the bug(s) to anyone you're not collaborating with. The learning benefits come from working through the problem—not from being handed the answer.

A few debugging tips:

* Choose small values for the dimension of the game board. It might be easier to see what's going on.
* Imagine you're the computer executing the code, step by step. Seek to fully understand it.
* As needed, add print statements to show where the "flow of execution" goes. For example, you can add a new first statement to a function (suppose it's called 'my\_func') of the form:

print('Now entering function my\_func')

* As needed, add statements to print out values that might help shed light on what's going on.

1. **[Turn in]** Submit three more "elevator pitch" ideas for your course project.

NWYH — Computer Science — Python — HW #11

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

1. Read chapters 12 and 13 of Think Python, 2nd ed. (Continuing with two chapters per week.)
2. **[Turn in]** Download the file **c2f.py.zip** from the course website, and unzip it to get **c2f.py**.

The script was written to demonstrate that functions can take lambdas as function arguments, and can also return lambdas as their return values. I added assert statements to ensure that the script worked as intended, but one of the assert statements fails. Please fix the script so that all of the assert statements succeed, and then submit the fixed script as an attachment.

1. **[Turn in]** Download the file **bank\_account\_transfer.py.zip** from the course website, and unzip it to get **bank\_account\_transfer.py**. Make the two modifications described in the file:
2. Define a function transfer, that transfers from the first argument (of type BankAccount) to the second argument (also of type BankAccount) the amount specified by the third argument (of type float).
3. Add a few lines to the end of the file to transfer $75.00 from the savings account to the checking account, then printing the values of both accounts.

Lastly, submit the resulting file as an attachment.

1. **[Turn in]**
   1. Browse through the draft of the Unit #3 slides that are available on the course website, in case they give you any possible course project ideas.
   2. Submit three final "elevator pitch" ideas for a course project. (This makes nine total.) These should present complete ideas using complete sentences.
2. Download the file **named\_bank\_account.py.zip**, and unzip it to get **named\_bank\_account.py**. This script is based on the class **BankAccount** that we saw in class earlier, with a few modifications:
3. Each account stores the name of its owner as an instance-level attribute.
4. A class-level dictionary called **accounts** is maintained that stores account owner names as dict keys, and the corresponding accounts themselves as dict values.
5. A function called **print\_balances** is added that uses this dictionary to print all the account balances.

Figure out what you think the output of the script would be, and then run it to confirm.

NWYH — Computer Science — Python — HW #12

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

1. Read chapters 12 and 13 of Think Python, 2nd ed. (The same two chapters assigned in HW #11.)
2. **[Turn in]** Download the file **bank\_account\_map\_filter.py.zip** from the course website, and unzip it to get **bank\_account\_map\_filter.py**. Make the one modification described in the file:
3. Use the **filter** function to get a list of **NamedBankAccount** objects that have at least $400.

Then submit the resulting file as an attachment.

1. **[Turn in]** Download the file **adventure.py.zip** from the course website, and unzip it to get **adventure.py**.
   1. Try running the program. Unfortunately, you'll see that it encounters an error as currently written.
   2. Look over the program (including the class and function names) to see how it works.
   3. Modify the **play** function in class **Player** to get to reach the winning room and win an awesome prize!
      * But do not modify the code elsewhere in the script—that's off-limits!
   4. Submit the winning version of the script as an attachment.
2. Download the file **fibonacci\_speed.py.zip**, and unzip it to get **fibonacci\_speed.py**. This script times how long it takes to compute 32 values of the Fibonacci function without using a cache, and 5000 values while using a cache. Run the script to see how the two times compare, and be thankful that many of the websites you visit use caches to make their web page load time bearable.