Assignment 2.2P

2022 Programming in Psychological Science

Contents

Assignment description

Work through the following 10 Python problems and 1 Python challenge program. This assignment is worth 2 points (0.1 points per regular problem and 1 point for the Python challenge program). You should also complete Assignment 2.1 for 8 points. The total of Assignments 2.1 (R) and your choice of either 2.2P (this Python assignment) or 2.2R (R challenges) are worth 10 points and 15% of your final course grade. For those students who are completely new to programming, we recommend attempting 2.2R (R challenges) instead of 2.2P (this Python assignment). You must work individually, but feel free to ask questions in class and on Slack!

You must submit the problems as 3 separate Python .py files. Submit all files in a .zip folder. The subheadings and problems make clear what problems belong in which files. All `import` lines should be at the beginning of the scripts / module. All answers must have comments and be labeled with the problem number.

Some problems have more than one solution. Best solutions are those that follow style guidelines (https://google.github.io/styleguide/pyguide.html), have clear variable names, and have comments so that other programmers (and your future self) can read and adapt the code.

problems__assignment2__2p.py (Problems Q2.2P.1 to Q2.2P.7)

Note that in the following problems we assume that the following imports are given at the beginning of the .py file.

```
import numpy as np
from time import time
```

Q2.2P.1 (0.1 points)

```
two_dice = np.random.choice(np.arange(1,7), 2, replace = True)
```

Consider a game of dice in which two dice are rolled and a score is determined from the roll.

The score is determined as follows: If the sum of the two dice is even, the score equals the sum of the dots (pips) multiplied by the smallest number of the two dice. For example, if you roll a 3 and a 5 then sum, 8, is even. The score is 3 * (3+5)=24.

If the sum of the two dice is odd, your score is the number of dots (pips) multiplied by -3. For example, if you roll a 2 and a 5 then sum, 7, is odd. The score is -3 * (2+5)=-21.

Write an **if-else statement** that calculates your score based on the values of the dice.

Q2.2P.2 (0.1 points)

Create the numpy array `np.array([3, 5, 7, 9, 11, 13, 15, 17])` using a **for loop**. Start with `double_plus = np.array([])` and fill the vector using the for loop. Hint: See `?np.append`

Q2.2P.3 (0.1 points)

```
grass = "green"


def color_it(color_me, grass_me):
    grass_me = grass
    color_me = "blue"
    grass = "blue"
    colorful_items = np.array([(color_me, grass_me)])
    return colorful_items


sky = "grey"
ground = "brown"
these_items = color_it(sky, ground)
print(these_items)
```

Remember the discussion about **global** and **local** variables within and outside **functions**. Does this code run in Python? Why or why not?

How is this different to R? Fix the code to make it run.

Q2.2P.4 (0.1 points)

Run the following code.

```
class MyClass:
    """A simple example class"""
    classnum = 12345

    def famous(self):
        return 'hello world'


new_stuff = MyClass()
new_stuff.classnum
new_stuff.famous()
```

What is a Python **class**? Briefly describe a Python **class** in a couple of sentences.

Read briefly Python **classes** here: https://docs.python.org/3/tutorial/classes.html

Now add a **definition** to the following Python **class** `ComplexNum` to return the phase angle in radians of the complex vector with the command `my_num.phase_angle()`. Try creating a few different complex numbers with this class.

```python
# Complex number class
class ComplexNum:
    """Creates a complex number"""
    numtype = 'complex'

    def __init__(self, realpart, imagpart):
        self.r = realpart
        self.i = imagpart

    def vec_length(self):
        return np.sqrt(self.r**2 + self.i**2)


my_num = ComplexNum(3.0, 4.0)
print(my_num)
print((my_num.r, my_num.i))
print(my_num.numtype)
print(my_num.vec_length())
```

Q2.2P.5 (0.1 points)

Write a definition `nthroot()` that calculates, by iteration, the nth root of a number using the Newton's method (https://en.wikipedia.org/wiki/Newton%27s_method). The definition should take as arguments, `number`, `n`, and a starting value `start`. The default starting value should be 1.

Hint: https://en.wikipedia.org/wiki/Newton%27s_method#Square_root

Q2.2P.6 (0.1 points)

Use `time()` from the module `time` to test how long `nthroot(4214,5)` takes compared to base Python code: `4214**(1/5)`. Which is faster?

Q2.2P.7 (0.1 points)

Quicksort is a famous algorithm. See https://en.wikipedia.org/wiki/Quicksort

Write the Quicksort algorithm in Python. This will likely require at least two **definitions**. See this pseudocode: https://textbooks.cs.ksu.edu/cc310/7-searching-and-sorting/19-quicksort-pseudocode/

sequences.py (Problems Q2.2P.8 to Q2.2P.10)

Q2.2P.8 (0.1 points)

Write a definition that gives the first `n` Fibonacci numbers as a **numpy array**. See https://en.wikipedia.org/wiki/Fibonacci_number

Use the following structure.

```python
def fibonacci(n):
    """
    Returns something...
    """
    # Your code here
    return result
```

Q2.2P.9 (0.1 points)

Change the Docstring `"Returns something..."` in your `fibonacci()` definition into an informative message and usage about your definition.

Now place your `fibonacci` definition into its own file called sequences.py. This is a called a Python module. Now import sequences.py into your IPython console with this line:

```
from sequences import fibonacci
```

Now type `?fibonacci`. Do you see your Docstring? Why might you want to keep a few definitions in their own .py file? Place your answer as a comment in sequences.py outside the definition.

More help: https://docs.python.org/3/tutorial/modules.html

Q2.2P.10 (0.1 points)

We might want to use the command `assert` to run tests for our program to make sure that it always returns the correct output, even when we change things in our program. `assert` returns an error if the logical is `False`. Running tests for large programs is very useful.

Add some assert programs at the end of your sequences.py file in the following format:

```
if __name__ == '__main__':
    assert fibonacci(1) == 0
    assert fibonacci(10)[-1] == 34
    print("Tests passed")
```

Write additional assert lines for more tests of your module.

hangman.py (Program Q.1.2P.11)

Q2.2P.11 (1 point)

Create a program that allows you to play Hangman in a terminal (e.g. the IPython console). Read more about the Hangman game if you are not already familiar with it here: https://en.wikipedia.org/wiki/Hangman_%28game%29.

Your Python program should pick a random word from a long list of words. The player in the Python console will guess letters in the words. The Hangman word list can be in any human language you choose (e.g. Nederlands, English, Deutsch, etc.)

We are not awarding points based on Hangman graphics. But graphics can be included in your program if you wish, either as text-based graphics in the terminal or in other graphics.