Assignment 2.2P

2024 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! Also, **using ChatGPT**, similar chatbots, LLMs, or other ANNs in any way for this assignment **is not allowed**. The reason for banning these tools is that we found they inhibit learning how to program. You are welcome to use these useful tools after the course is over (though check future UvA course restrictions).

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, 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 datetime import datetime
import time
import warnings
```

Q2.2P.1 (0.1 points)

Create an **if statement** that prints `Go to sleep!` if (your computer's system) time is between 00:00 (12am) and 05:00 (5am), else if (`elif`) the time is between 07:00 (7am) and 10:00 (10am) it prints `Eet je hagelslag!`, or otherwise (`else`) it prints `Gut gemacht!`. We recommend using the Python module `datetime` (importing with `from datetime import datetime`) that should have come with Anaconda Python, although you are not restricted to this module.

Q2.2P.2 (0.1 points)

Write a **for loop** that performs the following operations on a Numpy vector (named `numeric_vec`) of an arbitrary size to calculate a **weighted sum**. Then calculate a **weighted average** outside of the loop.

Even elements should be doubled and then added, while odd elements should just be added to the sum. The weighted average can be calculated by dividing the final weighted sum by the length of the `numeric_vec` times 1.5 outside of the loop.

For example, if

```
numeric_vec = np.random.uniform(low=0, high=100, size=4)
```

the loop would perform the following operations:

```
weight_sum = 2*0
weight_sum = weight_sum + numeric_vec[0]
weight_sum = weight_sum + 2*numeric_vec[1]
weight_sum = weight_sum + numeric_vec[2]
weight_sum = weight_sum + 2*numeric_vec[3]
weight_avg = weight_sum / (np.size(numeric_vec)*1.5)
```

Note that you can use a **if statement** within your for loop, but you don't have to if you're clever with Python.

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 **definitions** (compare to **functions** in R).

(a) Does this code run in Python? How is this different to R?

(b) Comment out one line in the definition so that it runs without error.

Q2.2P.4 (0.1 points)

(a) Write a **definition** that calculates all unique values of a Numpy vector without using the function `np.unique` nor any additional packages other than `numpy` and name it `special()`.

(b) The function should **raise an exception** if the input is not a Numpy ndarray. See this link for more information.

(c) The function should **give a warning** that `"All values are special!"` using the `warnings` module if all the elements in the input vector were already unique.

Q2.2P.5 (0.1 points)

(a) What is a `try` block in Python?

(b) Show examples of `try` block usage for your answer to Q2.2P.4 with a `try` block that prints the error but continues running after the error is printed. That is, make sure your entire .py script runs even when you have an example of the purposeful error from Q2.2P.4 (b).

Q2.2P.6 (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 about Python **classes** here.

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

```
# 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.7 (0.1 points, but a slight mathematical challenge)

Write a definition `nthpower()` that calculates, by iteration, the number to the nth power using Newton's method. The definition should take as arguments, `number`, `n`, and a starting value `start`. The default starting value should be 1. Note that you should test your definition with only small values of `n`. See also this more specific example.

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

Q2.2P.8 (0.1 points)

Write a definition that gives the first `n` prime numbers as a numpy array.

Use the following structure.

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

Q2.2P.9 (0.1 points)

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

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

```
from sequences import prime
```

Now type `?prime`. 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 is here.

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 prime(1) == 2
    assert prime(10)[-1] == 29
    print("Tests passed")
```

Write additional assert lines for more tests of your module.

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

Q2.2P.11 (1 point)

Create a program that allows you to play video poker in a console (e.g. the IPython console).

(a) Create a game of video poker as a Python definition. See this wikipedia page for more help on poker scenarios. Note that you do not need to know about other aspects of poker games. You are just simulating someone playing video poker **by themselves** at a casino, bar, or at home (not with other players).

That game code should do the following: it draws 5 playing cards with the **kind** of card and **suit** of the card as separate strings for each card, e.g. `"king"` and `"hearts"`. You can choose to ignore other cards like `"jokers"` (but you can use them in your game if you wish).

The following scenarios should take away money (or points – if you are against gambling):

- High card (none of the same **kind** of cards nor are the cards of the same suit nor all ordered)
- Only one pair (e.g. two cards of the same **kind**)
- Two pair (two cards of the same **kind** with two other cards of the same **kind**)

The following scenarios should give the player money:

- Three of the same kind
- Straight
- Flush
- Full House
- Straight Flush

You have a choice of how to award money/points. You can also award or take away money/points for other scenarios other than the ones listed. You have the freedom to print certain messages given certain scenarios. You also have a choice of inputs to your function (maybe a player's bet?, although no inputs are necessary to draw cards). The function should at least return the amount of Euros/points gained or lost.

(b) Generate a while code that runs your game multiple times until a player has reached a certain amount of money/points. See your answer to Q2.1.15.

(c) We are not awarding points based on graphics. But graphics can be included in your program if you wish, either as text-based graphics in the IPython console or in a figure.

(d) The solutions should be somewhat unique across students. Although the internal dynamics of the function may be similar.