NYU Tandon School of Engineering

CS-UY 1114 Spring 2023

Homework 04

Due: 11:59pm, Thursday, March 3rd, 2023

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 4 separate `.py` files named according to the following pattern: `hw4_q1.py`, `hw4_q2.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW4 - Q1 (etc.)
Date due: 2023-03-02, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

For this specific assignment, you may use everything we have learned up to, **and including**, variables, types, mathematical and boolean expressions, user IO (i.e. `print()` and `input()`), number systems, and the `math` / `random` modules, selection statements (i.e. `if`, `elif`, `else`), and `for`- and `while`-loops. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, user-defined functions (*except* for `main()` if your instructor has covered it during lecture), string methods, file i/o, exception handling, dictionaries, lists, tuples, and/or object-oriented programming.

Failure to abide by any of these instructions will make your submission subject to point deductions.

Problems

1. **(s)rewoP ehT toG ev'I (`hw4_q1.py`)**
2. **Tamako Market (`hw4_q2.py`)**
3. **Must be funny in the rich man's world (`hw4_q3.py`)**
4. **A Little Imagination Goes A Long Way In Fez (`hw4_q4.py`)**

Question 1: *(s)rewoP ehT toG ev'I*

The task is similar to last week's lab question, which was to print the powers of the a user-input number up until a certain limit (which the user will also determine). Write a problem that will do this, but will only print the ***even*** (i.e. 2, 4, 6, etc.) powers, and will do so ***backwards***.

For example:

```
Please enter a positive integer to serve as the base: 7
Please enter a positive integer to serve as the highest power: 13

7 ^ 12 = 13841287201
7 ^ 10 = 282475249
7 ^ 8 = 5764801
7 ^ 6 = 117649
7 ^ 4 = 2401
7 ^ 2 = 49
7 ^ 0 = 1
```

Here's the kick, though: you may *not* assume that the user will enter whole-number values; they can also enter floating-point values. Your program should keep asking the user to enter both of these values *until* they are **positive, non-zero, whole-number values**:

```
Please enter a positive integer to serve as the base: 4.5
Invalid value for the base (4.5). Please enter a positive integer to serve as
the base: -5
Invalid value for the base (-5.0). Please enter a positive integer to serve as
the base: 4

Please enter a positive integer to serve as the highest power: -4
Invalid value for the base (-4.0). Please enter a positive integer to serve as
the base: 4.0

4 ^ 4 = 256
4 ^ 2 = 16
4 ^ 0 = 1
```

The format of your user input prompts and warning messages does not need to look the same as ours, but when you print out the powers, it must be in a `base ^ power = answer` format.

Question 2: *Tamako Market*

In this program, we're going to use Python to calculate how many ***mochi*** we can make using a certain amount of ingredients. A basic recipe for a batch of 24 pieces of ***daifuku mochi*** calls for:

- 3 cups of sweet rice flour (*mochiko*)
- 1.5 cups of sugar
- 2 cups of cornstarch
- 1 cup of red bean paste (***anko***)

Your program will work is as follows:

1. The user will enter a certain amount of *mochiko*, sugar, cornstarch, and *anko* in ***grams***.
2. The program will then convert those gram amounts to cups (1 cup = 220g).
3. The program will then calculate how many batches of *daifuku mochi* can be made with this quantity of ingredients.

A sample program execution could look like this:

```
Enter an amount (g) of mochiko: 2500
Enter an amount (g) of sugar: 3400
Enter an amount (g) of cornstarch: 5000
Enter an amount (g) of anko: 3200
With this amount of ingredients, you can make 3 batch(es) of 24 mochi.
```

For this problem, you can assume that the user will always enter positive numerical values.

Question 3: *Must be funny in the rich man's world*

Let's say that you are tasked with writing the final money counter program for a Monopoly-like video game. If you don't know what **Monopoly** is, don't worry; all you need to know is that at the end of each game of Monopoly game, all of the players will have a certain amount of money, and the person with the most amount of money wins (kind of a **terrible concept**).

So, your task is to write a program that

1. Asks the user how many players will be playing in this round of Monopoly. *This* value *must* be a positive, non-zero integer, but you can assume that the user will never enter a float value.
2. Once they do so, each player will enter the values of each of their properties/assets. You can assume that *these* will always be positive numerical values or the characters `"DONE"` when they are finished.
3. Once a player enters all the values, the game will print out the sum of the assets' values.
4. The program repeats steps 2 and 3 until all players have been accounted for.
5. At the very end, the program will print a congratulatory message to the winner. You don't have to worry about two players getting the same score.

Here's a sample execution. Yours doesn't need to look exactly like hours, but it must interact with, and show the same information to, the users:

```
How many players played this round? -5
Invalid input. How many players played this round? 3
Enter the value of a property/asset, or DONE to finish: 0.45
Enter the value of a property/asset, or DONE to finish: 23.56
Enter the value of a property/asset, or DONE to finish: DONE
Player 1 has 24.01 dollars.
Enter the value of a property/asset, or DONE to finish: 3.78
Enter the value of a property/asset, or DONE to finish: 3000
Enter the value of a property/asset, or DONE to finish: 34.87
Enter the value of a property/asset, or DONE to finish: 3
Enter the value of a property/asset, or DONE to finish: DONE
Player 2 has 3041.65 dollars.
Enter the value of a property/asset, or DONE to finish: 40
Enter the value of a property/asset, or DONE to finish: DONE
```

```
Player 3 has 40.0 dollars.
Congratulations, player 2! You won with $3041.65!
```

Problem 4: *A Little Imagination Goes A Long Way In Fez*

Note: This program will make use a module we don't officially cover in class called **Turtle Graphics, or `turtle`**. In some IDEs, such as PyCharm and VS Code, `turtle` sometimes doesn't work quite as it should (if at all). If this happens to you, feel free to use IDLE for this problem, or **this website**. If you do use the site, though, make sure to erase the starter code they provide and to save often to avoid potentially losing your work. All screenshots of sample behaviour below were taken using IDLE.

Turtle graphics is, basically, easy to use. The only two functions you need from it are `forward(distance)` and `left(angle)` which, as you can imagine, make `turtle` go forward by a certain `distance` and turn left by a certain `angle`. The following code, for example:

```python
import turtle

turtle.forward(100)
turtle.left(34.4)
turtle.forward(234)
```

Results in the following `turtle` window:

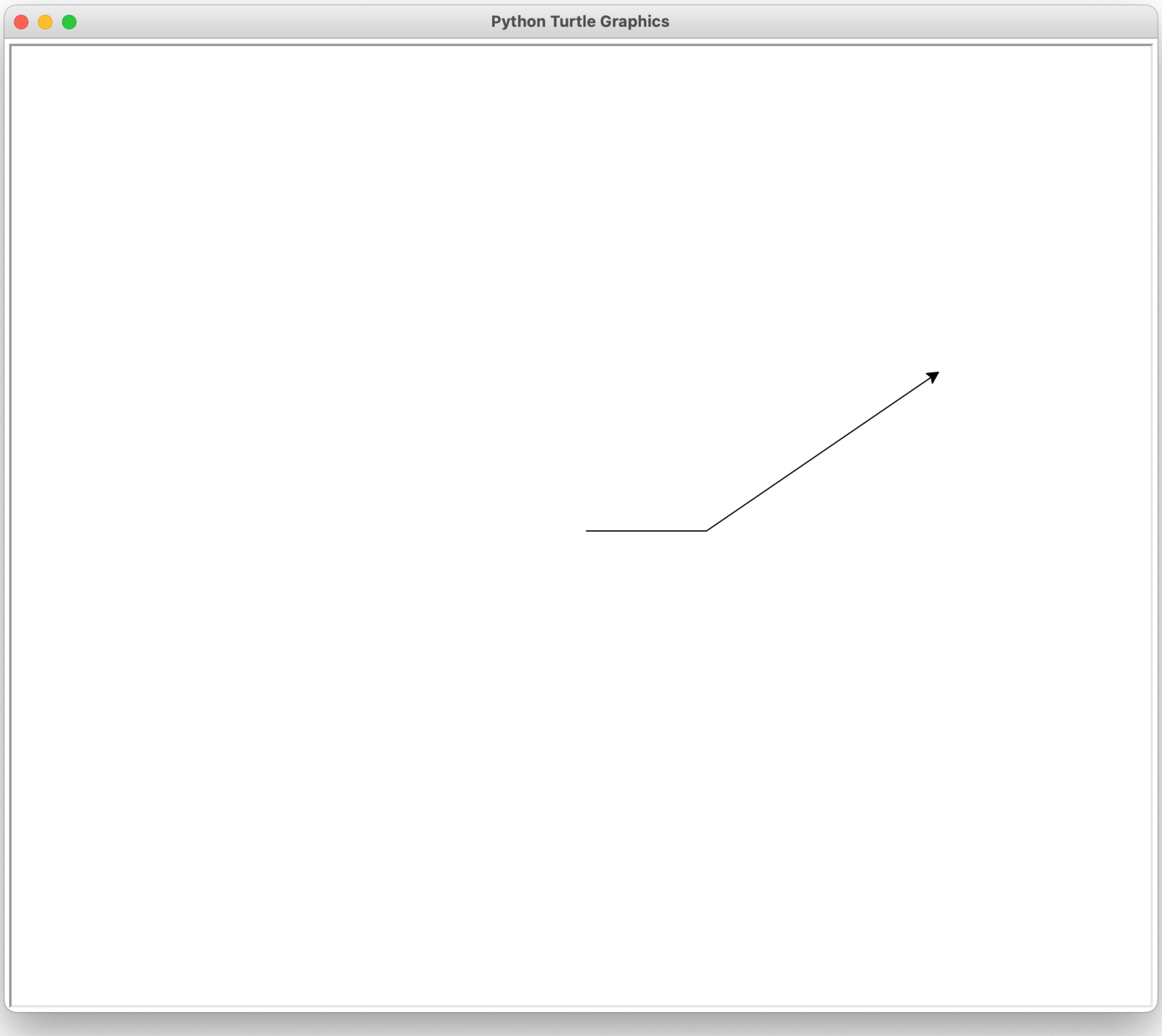

Figure 1: `turtle` moving 100 pixels in its starting orientation ("east"), turning 34.4 degrees counter-clockwise, and then moving 234 pixels forward.

That's all you need to know. Now, for the problem.

Moroccan mosaic, a.k.a. ***Zellij* (الزليج)**, is a form of Islamic art and one of the main characteristics of Moroccan architecture. It consists of geometrically patterned mosaics, used to ornament walls, ceilings, fountains, floors, pools and tables. Each mosaic is a tile-work made from individually chiseled geometric tiles set into a plaster base.

We can draw simple ones using `turtle`:

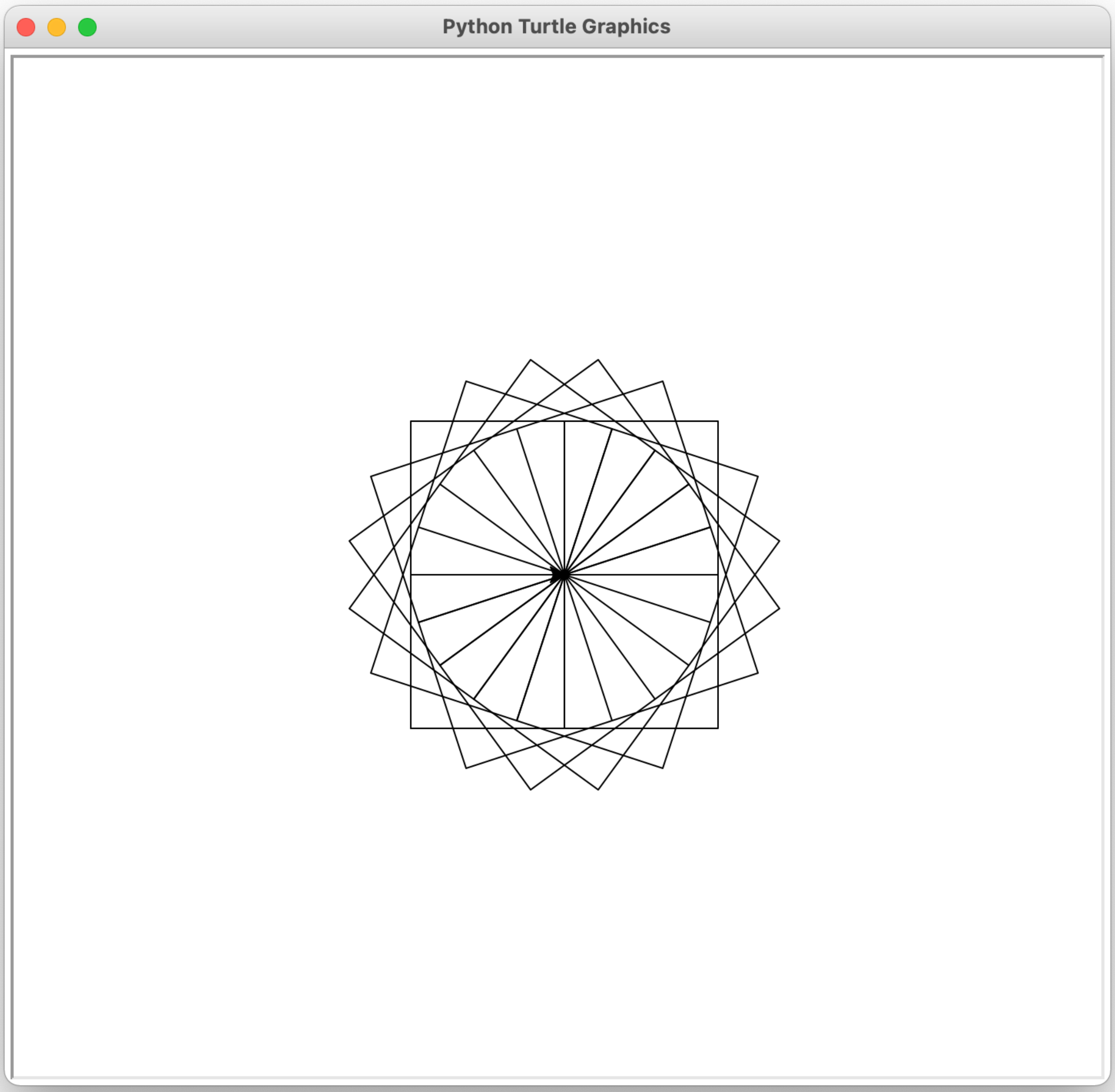

Figure 2: An example of drawing a mosaic using *20 squares*.

We achieve this effect by telling `turtle` to draw a regular polygon (e.g. pentagon, hexagon, etc.) and repeat and rotate this shape several times to create a circular pattern/mosaic.

Write a program that will ask the user for an integer representing the *shape of the polygon* you would like to use for your pattern.

Here's an example using `7` as input:

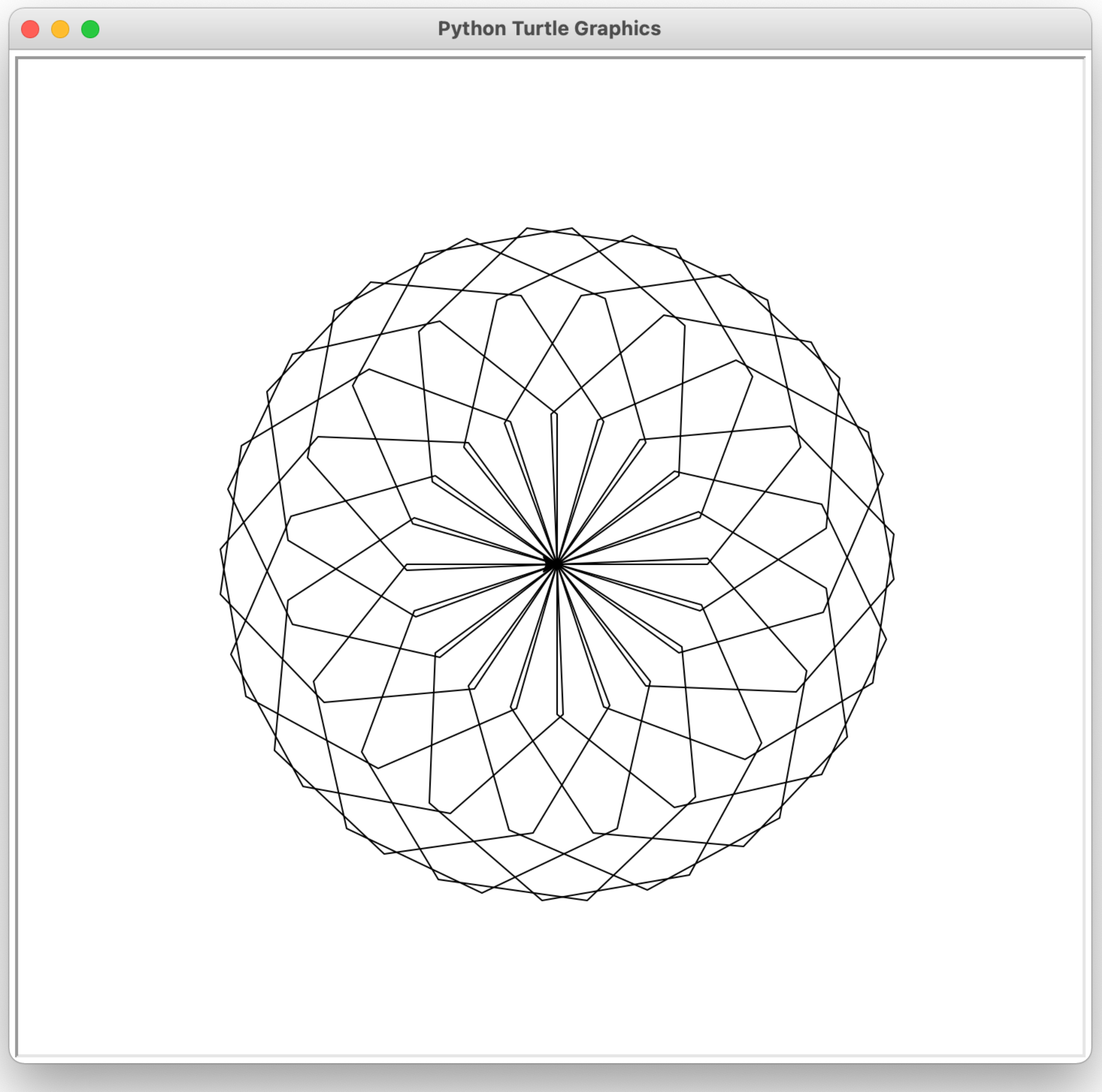

Figure 3: An example of drawing a mosaic using *20 heptagons*.

A couple of things to help you here:

- You may assume that the final number of polygons needed to draw a *Zellij* is set to 20.
- The inner angle of an `n`-polygon is equal to `360 / n`.
- The angle between two polygons is equal to 360 divided by the final number of polygons.