Homework 2

Assignment Focus: Functions and Testing

Please review this week's Developer notes (in this module) on writing functions, testing and debugging before starting your homework.

As always, to submit your solution to Gradescope. You can either drag-n-drop all of your files as a set OR compress all files together into one .zip file, and upload that single zip file. You may submit multiple times right up until the deadline; we will grade only the most recent submission.

You are permitted three "late day" tokens in the semester; each one allows you to submit a homework assignment up to 48 hours late but you must contact Prof. Keith before the deadline if you want to use a token.

This is an introductory course, and we want to make sure that everyone has a solid understanding of the fundamentals, so we'll often rule some Python tools in or out. For this homework, **do not** use conditionals, lists, tuples, loops, or dictionaries -- we haven't gotten there yet, and you don't need them!

For the code portion of this assignment, you are responsible for adhering to our style guide sections CS1-CS5.

Written Component (20% of your homework score)

- Submit file: written.txt

Please open a plaintext file (you can do this in IDLE or any plaintext editor you like, such as TextEdit or NotePad) and type out your answers to the questions below. You can type your answers into Python to confirm, but answer the questions first!

Written #1

For each of the functions below -- identified by name, parameters, and return type -- write a line of Python code that demonstrates how you would call that function. Pass literals as your arguments to the function (no need to define variables); these literals can have any values you like. If the function returns something, save the result in a variable.

1A

```
Name: print_formatted
```

Parameters: one string
Return type: nothing (void)

1B

```
Name: get_max
```

Parameters: two integers
Return type: int

1C

```
Name: log_base_two
```

Parameter: int
Return type: float

Written #2

In the following Python snippet, line 3 will never execute. In your own words, explain why not.

```
1  def divide(x, y):
2      return x / y
3      x = x + 5
4
5  def main():
6      div = divide(18, 3)
7      print('The answer is', div)
8
9  main()
```

Written #3

What does each of the following Python programs print to the terminal?

3A

```
1  def f1(m):
2    m = m + 1
3    print(m)
4
5  def main():
6    m = 18
```

```
7       f1(m)
8       print(m)
9
10  main()
```

3B

```
1   def f1(m):
2       m = m + 1
3       print(m)
4
5   def f2(n):
6       print(n)
7       n = n + 1
8       return n
9
10  def main():
11      m = 18
12      f1(m)
13      m = f2(m)
14      print(m)
15
16  main()
```

Written #4

Below are two functions that do roughly the same thing. If you have to choose one, which function is better than the other? In your own words, describe why.

Function #1	*Function #2*
def diff_squared(x1, x2): **diff = x1 - x2** **sq = diff ** 2** **return sq**	**def diff_squared(x1, x2):** **diff = x1 - x2** **sq = diff ** 2** **print("Diff squared is", sq)**

Written #5

5A:

Given this function you saw in question #2:

```
def divide(x, y):
    return x / y
    x = x + 5
```

Rewrite the header of this function (its signature) to use Python annotations to indicate parameter types.

5B:

Python annotations are optional and many "old school" Pythoners don't use them. In your own words, give one reason (or benefit) to using annotations for function definitions

Programming Component (80% of your homework grade)

Note: Only ONE (1) flowchart is due this week – submit one for Programming #2. It's suggested you continue practicing/using flowcharts for the other assignments (for your own benefit) but do not submit them for Prog #1 or #3.

Code Structure and Style

A percentage of your score for every homework is for writing good, clear, readable code. For HW2, focus on writing good functions. Read the style guide and remember our guidelines:

- A function has ONE JOB
- It has a descriptive name, ideally a verb
- It has docstring comments at the beginning with parameters, return type, and description if needed (optionally include the function's name in the docstring)

Programming #1 (25% of this HW)

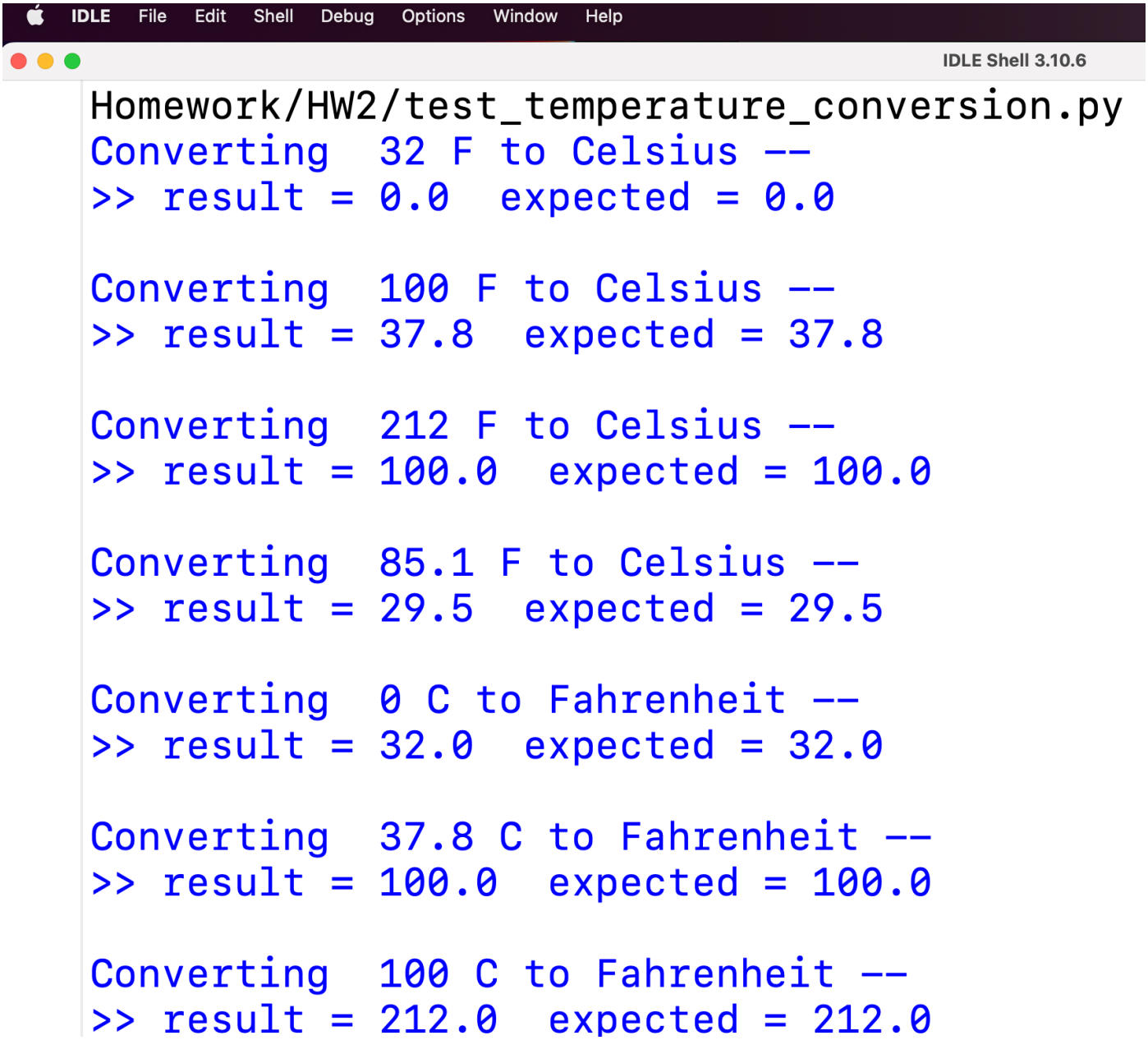
- Filename: test_temperature_conversion.py
- Starter code: temperature_conversion.py

The United States is one of the last countries to still use the imperial system and Fahrenheit for measuring temperature. 🤷🏾
One software conversion error between US and metric measurements sent a $125 million NASA probe to its fiery death in Mars' atmosphere .

We're going to avoid situations like that by having you practice testing, testing, testing!

We've included some "starter code" that you must download to complete this part of the assignment. So do that first: download the **temperature_conversion.py** file. This starter code includes two functions that convert between Celsius and Fahrenheit. Your job is to write Python code to test these functions. You should write your test functions AND a main() function in a file you create called **test_temperature_conversion.py**. When we run your **main()**, we should see all of your tests, the expected result for each one, and the actual result for each one.
Note: Be sure to include the **if __name__ == "__main__":** trick you've learned in this course to "protect" your main() from running inadvertently!

You have flexibility with regards to your output since we won't be using the Gradescope autograder to examine your output text.
However, we will be manually inspecting your output, so **be sure your output is readable and clean**. Ensure that your output is formatted to 1-decimal-place of precision. Here's an example of part of our test suite running (this is a small snippet, not the entire run):



...etc.
You should write tests for both of the starter-code functions provided.

Structuring your code:

- You have little-to-no repeated code, because you've so wisely put anything that gets repeated into functions.
- We won't be using the auto-grader for this part of the assignment. **To earn full credit for correctness, make sure your output is clean, clear and easy to read for visual verification of results**

Programming #2 (30% of this HW)

- Filename: windchill.py
- Starter code: temperature_conversion.py

Now we'll build on the verification and validation you did in programming #1. You've already tested the code in the temperature_conversion package, so now you'll actually **use** those functions here.

When the wind blows in cold weather, the air feels even colder because the movement of the air increases the rate of cooling for warm objects. Meteorologists call this effect the "wind chill index". In effect the wind chill index is the temperature it "feels like" when the wind is blowing. For example, the absolute temperature may be 22F but because of the windchill, it feels like 11F

In 2001, Canada, the USA and the U.K adopted the following formula for computing wind chill, where **T** is the air temperature in Celsius and **V** is the wind speed in kilometers-per-hour

Windchill Index = $13.12 + 0.6215T - 11.37*V^{0.16} + 0.3965*T*V^{0.16}$

Do This: Create your solution file called **windchill.py** and write a function called: calculate_windchill(). This function takes two parameters: **temperature** (Fahrenheit) and **speed** (Miles-per-hour). The signature of your function should be similar to this: calculate_windchill(temperature, speed). Your function MUST use the windchill index formula given above to compute and **return** (answer back) the windchill index based on the input parameter values, and that answer must be in imperial units. See Note 2 below.
Your docString for this function must also include at least two docTest examples - see Reminder 1 below.

Note: there is an alternate formula for calculating the windchill index, using

imperial values instead of metric values. Your solution will **NOT** receive full credit if you use that (imperial) formula instead of the metric version given above! This means your code is obligated to convert from imperial to metric values (and you should be using the functions we provided, since you tested them in part 1 of this homework!)

Note 2: the formula above (which you must use results in the metric windchill temperature (Celsius). Your solution must return the index in imperial units (Fahrenheit). Be sure to convert your answer back to Fahrenheit after the calculation is performed!

We will be using the auto-grader for this part of the assignment so your function must match the specs precisely with respect to name, number of parameters, and order of parameters. Here is the signature of the function, with an example docString describing what it should do:

calculate_windchill(temperature, speed):

```
'''
Function: calculate_windchill
    Calculates windchill based on international formula (Metric)
Parameters:
    temperature in Fahrenheit
    speed in miles per hour
Returns: windchill index (floating point value) based on applied formula
Require: temp/speed in metric units
Ensure: metric -> imperial unit conversions prior to calculation
'''
```

Reminder 1: Your submission must enhance the docString I've given you for calculate_windchill() (immediately above this sentence) with at least two (2) *docTest* statements that we can use with the docTest framework to validate the documentation and proper use of this function.

Additional Note: You do NOT need to provide a main() for this part of the assignment. You may write other "helper" functions if you wish (make sure you import any extra files you create AND include those files in your homework submission), but you must provide the calculate_windchill() function precisely as specified here.

*Additional Note: You must use the temperature_conversion functions provided (you tested them thoroughly, right?). Hardcoding conversion values in your solution will result in marks being deducted from your grade. The one exception to this is the MPH to KPH (miles-per-hour to kilometers-per-hour) factor. You may look this up and use a constant for this value (**1.61** is the conversion value if you need it)*

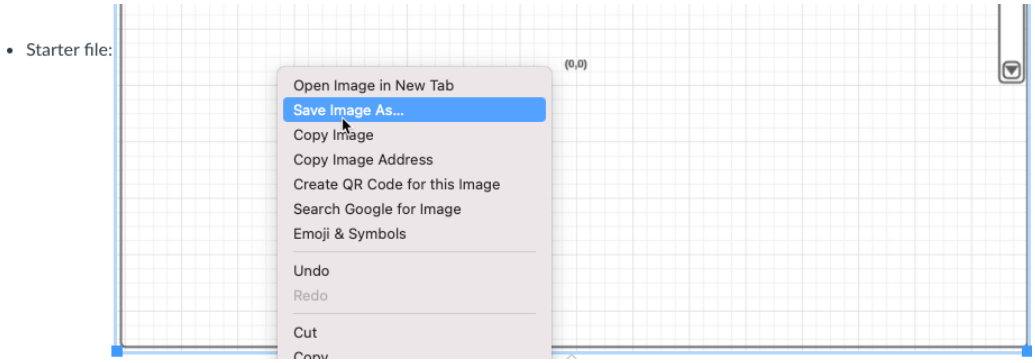
Why does the USA still use Imperial measures?: https://www.vox.com/2015/2/16/8031177/america-fahrenheit

Your function should handle any reasonable, legal arguments we use when our auto-grader calls it

Programming #3 (25% of this HW)

- Filenames: align_draw.py (and any other files you need)
- Starter file:

***Note**: If you're having trouble downloading the starter file (it's the graphic image above called shape_window.png) right-click on the image and select "Save image As..." to save it on your local device.*



All programmers need to be able to get the hang of a new library or learn someone else's code. With this last HW2 problem, we'll get some practice with that -- with a Python graphics library called Turtle.

Use the Python Turtle library to render the supplied background image that looks a bit like an application window. Read up on the Turtle documentation

online here: Python Turtle . Here are a few things you might want to look up:

- How to initialize a screen to a given size
- How to use an image as your background
- How to move the turtle to a given (x, y) position without drawing a line on the screen
- How to draw a circle, how to draw a square (we covered some of this in class)
- How to draw filled-in shapes with begin_fill()
- How to clear the screen after drawing

In Python, the center of the screen is at (0, 0) by default.

Your program should do the following:

1. Draw a square with a length of 80. The center of your square should be at (0, 0). Use any color you want; in the example screen captures here, I'm using blue.
2. Draw an inscribed circle in the square that was drawn in step 1. Use a different color for your circle. I used magenta but pick any color you like – other than what you used for your square. *Due to the way Turtle draws lines and circles, you'll likely need to experiment with your starting coordinates for your circle draw.*
3. Prompt the user for a new (x, y) coordinate for the square, and a new (x, y) coordinate for the circle.
4. "Move" the shapes to their new positions by erasing the original square and circle, and drawing them in the location specified by the user. The redrawn shapes should be solid-filled with the color of your choice (the shapes' center-points are specified by the information input by the user).

Define any functions you need to make this all work. The "app window" is screen capture I saved to a png file, so your drawings might be a pixel or two shifted. That's okay as long as the placement and movement seem

reasonable from the (0, 0) origin you start at.

You can assume the user gives you “good data”: values all within the Turtle coordinate system between 200 and -200.

Notes:

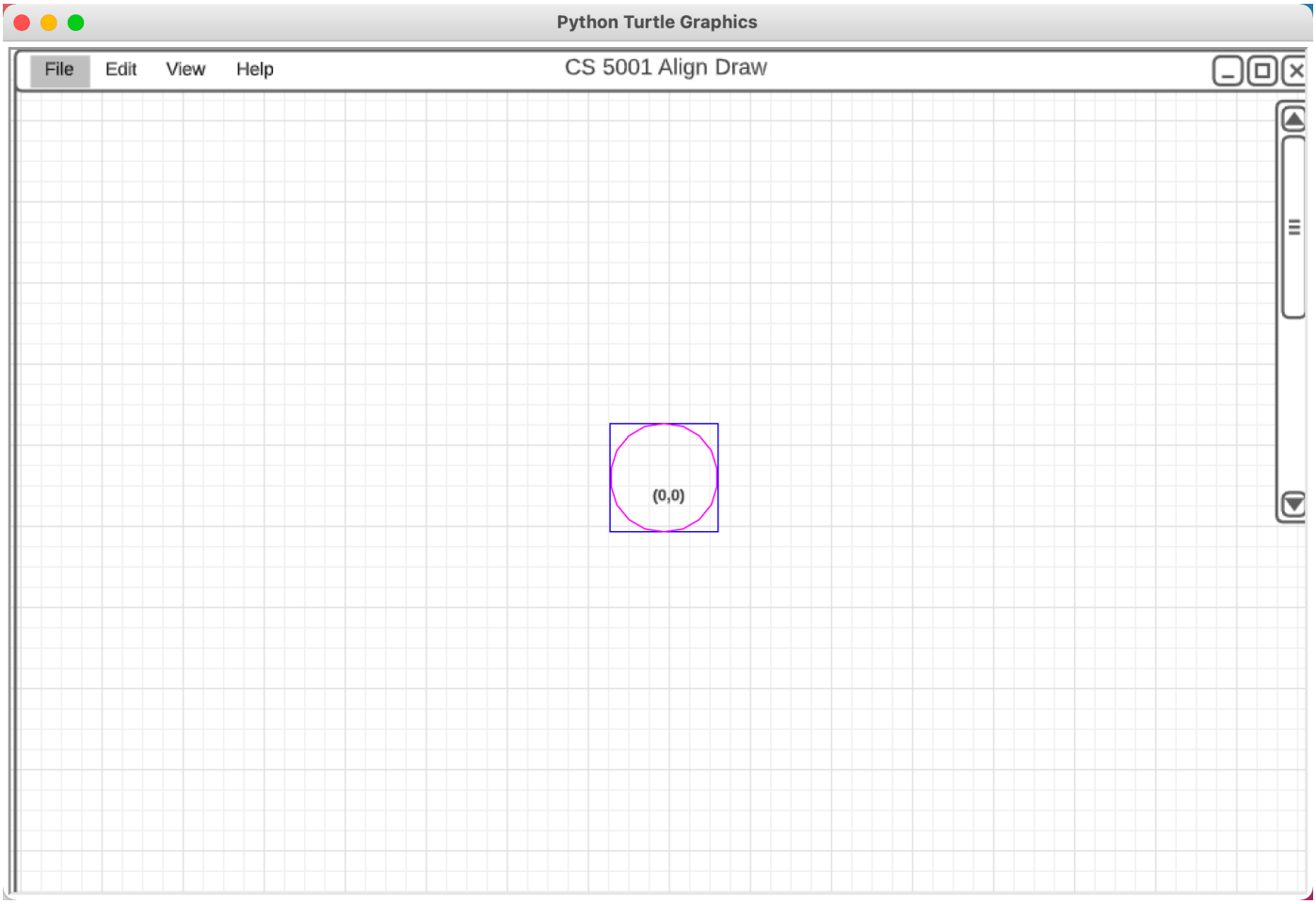
- Consider the given .png image of the application window is 970x635 pixels, and that should be the starting size of your Turtle screen.
- If you don’t remember your geometry, an inscribed circle looks like this, and the diameter is equal to the length of the square’s side.

Tips:

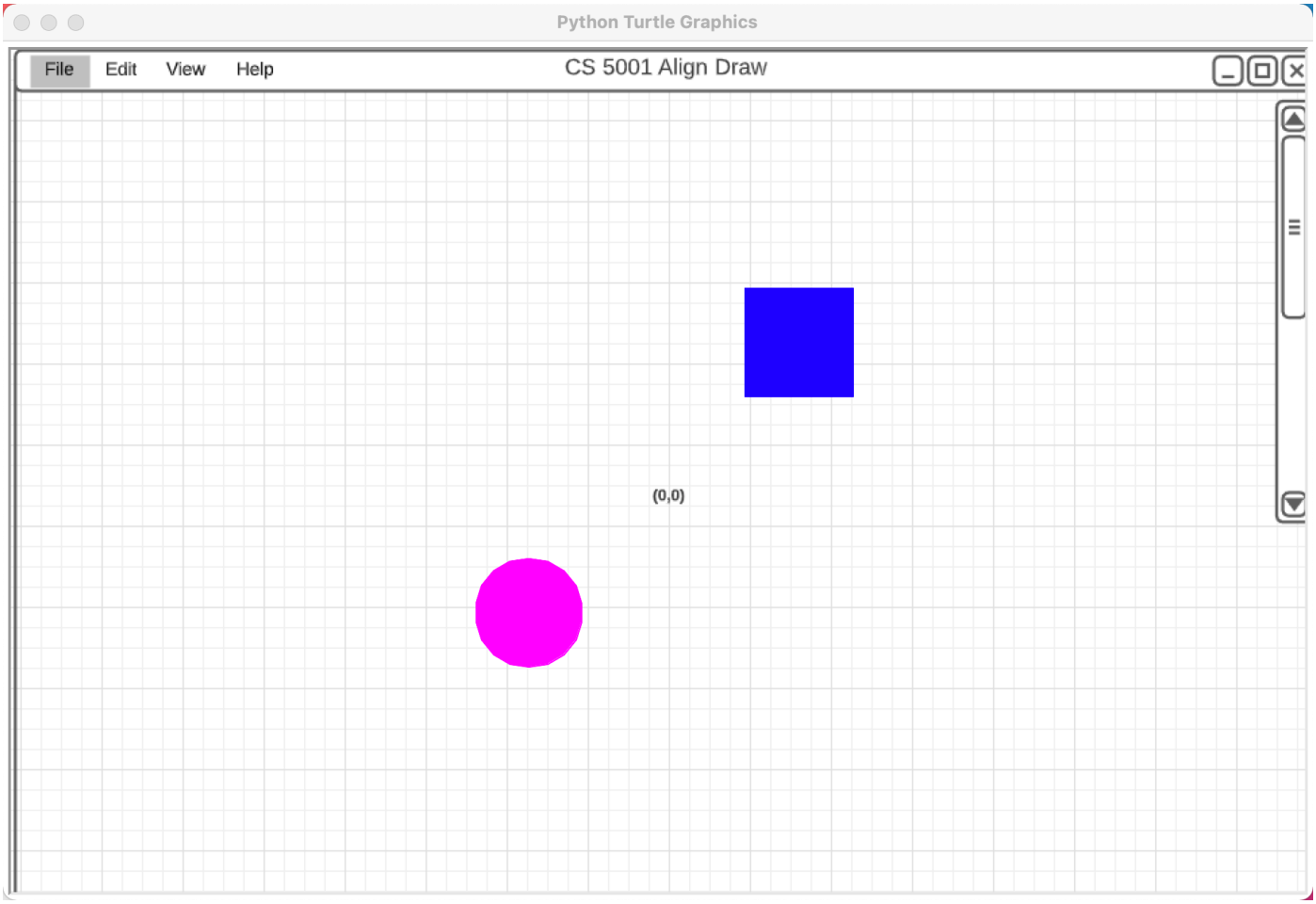
- It’s very difficult to test graphics-related functions without explicit test tools (screen-grabbers, functional test tools, etc.), so your most important job is to ensure your turtle movement and drawing is as consistent as possible. Of note, when you move the turtle, it has a directional heading (N/S/E/W). Take care to set the heading consistently to ensure repeatable drawing of the shapes.
- You can trust the user to give you good information, and to stay within the ±200 (the default bounds of the drawing canvas coordinates)

Example

- Here’s what my “app window” looks like after drawing the square and inscribed circle



- After being asked for new coordinates, here's what my "app window" looks like after the shapes have "moved"



Pro-tips:

- Create a separate “driver” file for your main() function; have all your other functions reside in a separate file
- Try to keep your main() small – perhaps 2 or 3 lines of code.