Project #4 – Common Math Formulas

1 Objective

When finished with this project, you'll be able to…

- Pass data to, and return data from, functions (including returning more than one value)
- Implement a module to create a function library[1]

2 Problem Overview

Assume a young family member is studying common math formulas, using this page as a starting point. You want to write some code to help them check their work. *An aside: some of the questions under the Circle topic don't rely on the provided circle equation; we'll focus on the exact equation, though.*

3 User Interface

3.1 Input and Output

The program will ask for the raw data on which to perform calculations. Output should be as shown below. *Note that the calculated numbers shown are intentionally incorrect; you'll need to figure out what's correct. Also note that numbers are shown with 4 digits after the decimal point.*

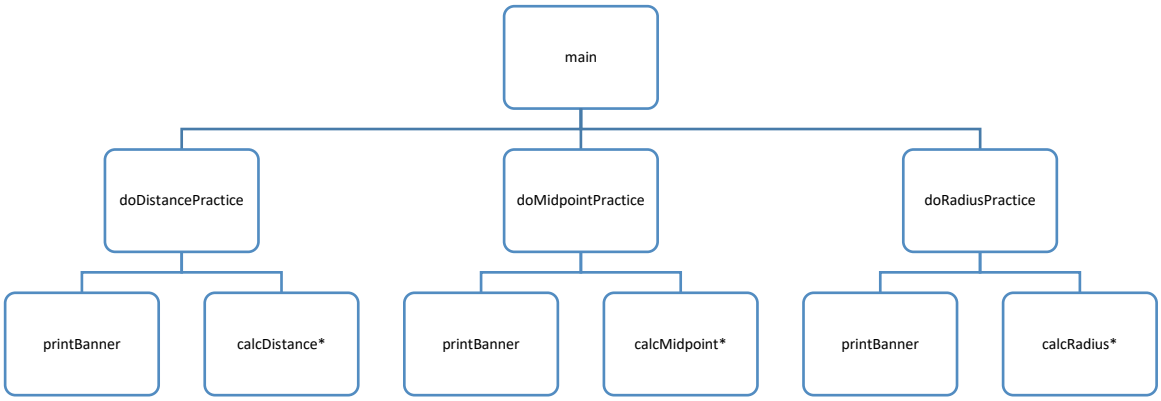
```
----------------------------------------
         DISTANCE PRACTICE
----------------------------------------
Enter x1: 1.3
Enter y1: 4.5
Enter x2: 8.9
Enter y2: 11.3
Distance = 8.1234

----------------------------------------
         MIDPOINT PRACTICE
----------------------------------------
Enter x1: 7.8
Enter y1: 1.4
Enter x2: 15.6
Enter y2: 18.3
Midpoint x = 13.2345
Midpoint y = 3.3456

----------------------------------------
         RADIUS PRACTICE
----------------------------------------
Enter center x: 4.5
Enter center y: 7.8
Enter point x: 9.3
Enter point y: 11.5
Radius = 17.4567

That wraps up practice time.  See you next time!
```

[1] This is our first exploration of Supplier Code, which is code that doesn't interact with directly with the user, but instead *supplies services for other code to use*. Most code in the world is supplier code. Different rules apply in supplier code, including no display output on the console and no direct interaction with the user.

4 Functions

4.1 Call Hierarchy

Asterisked functions must be defined in **formulas.py**; this entire module is supplier code. Other functions are defined in the main project file, **proj4.py**, which is client-facing code where all input and display occurs.



4.2 Function Specifications

4.2.1 Descriptions

Here's a brief description of what each function should do, and what file it should live in:

Function	Description	File
main	Calls the do* functions and controls spacing in between them	proj4.py
printBanner	Prints the number of hyphens specified in a global constant called BANNER_WIDTH, then prints the title specified (one tab in), then prints another row of hyphens	proj4.py
do*	Manage practices for their portion. Call for the printing of an appropriate banner, ask for pertinent data, and display results	proj4.py
calc*	Receive pertinent parameters, do the requested calculation, and return value(s) resulting from these calculations	formulas.py

Important: no user input or output must be done in functions contained within the formulas.py file.

4.2.2 Parameters and Return Values

You need to determine the parameters and return values for each function. But here are some rules you'll need to follow in your quest:

- There may be *no* global variables; you must pass data to functions to communicate it. Global constants are okay (e.g., BANNER_WIDTH described above).
- All functions that calculate mathematical results must be in a module (a separate file) called formulas.py. The main program will need to import that module and use it.
- Name parameters and arguments appropriately; their names aren't required to match; it may make sense for these to different in distinct contexts.

4.3 Calculations

Calculations are as shown on the linked web page (here, as a reminder).

5 Code Specifications

- At the *bottom* of the program, you should have a call to main(). This should be the only line of code outside a function definition (imports don't count).
- Include header comments at the top of each file. Include your name, the date, and a brief description of what the program does.
- Include comments for each section saying what's going on in the lines of code below, e.g., `# calculate distance between two points`
- Use comments elsewhere as you think they help guide the reader. Don't overdo, though; not every line needs a comment; think about describing a block of related code.
- Use blank lines to separate sections and provide visual "breathing room."
- Use descriptive variable names.
- For formatting, use format() only, not the format method, not "f strings," etc.
- Do not use lists or other tech we haven't covered; what has been presented in class is sufficient.

6 Hints

- Use exactly the call hierarchy shown above. Do not "chain" functions to force execution order.
- Carefully consider parameters for each function. Think about what that function needs from outside of itself (if anything) to do its work, or what it needs that it must pass on to functions it calls.
- One of the functions you'll use returns more than one value; you will need to make use of Python's ability to return multiple values via tuples.
- You must submit two .py files this time; make sure both files are uploaded in a single submission.

7 Testing

- Develop an appropriate number of test cases. Calculate results using some other method (e.g., by hand, using Excel, etc.), then confirm the program yields the same results.
- Test a few error cases and see what happens. Realize at this stage you're not equipped to solve all problems you'd like to; keep notes about what you'd like to do once you learn more.
- Remember that testing is not just about calculations; UI needs testing as well.
- Document your testing and results in comments at the bottom of the program as shown below.

8 Extra Credit

Create a PyUnit test script called `TestFormulas.py`. In it, create an automated test case for each of the functions in the formulas.py file. Verify that the functions produce correct results to 4 digits after the decimal point. Use the materials in the Canvas Resource module (at the top), including the Introduction to Software Testing document, and the two linked videos that demonstrate how to construct automated tests. To ensure correctness, calculate results independently of your code, so you don't accidentally encode the same bad logic in both the functions and the tests.

9 Summary

At the bottom of your program, add comments that answer these questions:

- How did you approach this assignment? Where did you get stuck, and how did you get unstuck?
- How did you test your program? What doesn't work as you'd like, perhaps things that you'd like to fix as you learn more?
- What did you learn from this assignment? What will you do differently on the next project?

10 Grading Matrix

Area	**Pct**
Call hierarchy followed	10
Functions implemented	20
Parameter passing, returns	20
Module creation/usage	10
Output correctness	10
Output formatting	10
Test cases	10
Comments, variable names, white space	5
Summary report	5
Extra credit	5
Total	**105**