# Project #4 – 2D & 3D Geometry Calculator

## 1 Problem Overview

In a geometry class, you are studying interesting formulas for circles, squares, spheres, and cubes. You are writing a program to encode those formulas and generate accurate results.

## 2 User Interface

### 2.1 Input

The program asks the user for a size (which will be used for radius and side length, depending on the shape), and the unit of measure being used, e.g.,

```
Enter size for side and radius: 5
Enter unit of measure, e.g., inches: inches
```

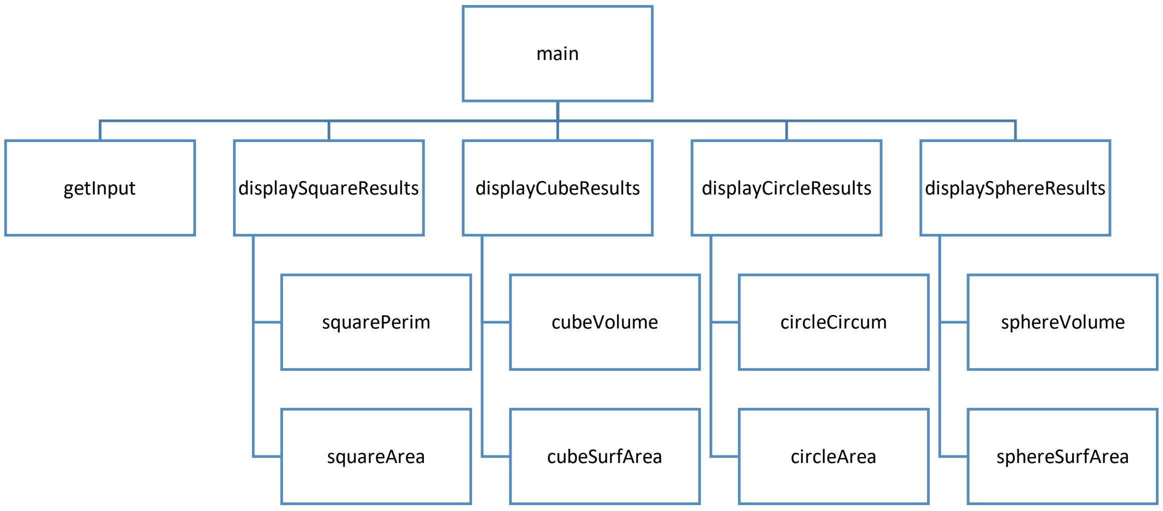
### 2.2 Output

Displays an attractive and aligned report with all the results, as shown below. Note that the numbers here are merely samples and are *intentionally incorrect*; you will need to figure out what is right here!

```
--------------------------------------------
Square Calculations
--------------------------------------------
Perimeter:            9.85 linear inches
Area:                17.91 square inches

--------------------------------------------
Cube Calculations
--------------------------------------------
Volume:             514.99 cubic inches
Surface area:        75.32 square inches

--------------------------------------------
Circle Calculations
--------------------------------------------
Circumference:       13.83 linear inches
Area:                29.22 square inches

--------------------------------------------
Sphere Calculations
--------------------------------------------
Volume:             146.25 cubic inches
Surface area:        83.11 square inches
```

# 3 Functions

## 3.1 Call Hierarchy



## 3.2 Function Specifications

### 3.2.1 Descriptions

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

| Function | Description | File |
|---|---|---|
| **main** | Consists of a series of function calls and little else | proj4.py |
| **getInput** | **Asks** the user for a size and a unit of measure, returns those values | proj4.py |
| **display*** | **Prints** a header line (e.g., "Square Calculations") and the results of calculations for that object (e.g., all corresponding Square calculations) | proj4.py |
| **square*, cube*, circle*, sphere*** | **Calculates** and returns the specific results, e.g., CubeVolume calculates the volume of a cube with the specified side length | p4geom.py |

### 3.2.2 Parameters and Return Values

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

- There may be *no* global variables; you must pass data to functions to communicate it.
- All functions that calculate geometry results must be in a module (a separate file) called p4geom.py. The main program will need to import that module and use it.
- The p4geom.py module must use the math module's pi variable. That means p4geom.py will need to import the math module.
- Watch the names of parameters and arguments; they should not always match as the sender and receiver may interpret the data in slightly different ways. For example, getInput gathers a generic *size* from the user, but circleArea clearly wants a circle's *radius* as a parameter.

### 3.3 Calculations

The geometry calculations are standard ones:

- Circle: $C = \pi d$ and $A = \pi r^2$
- Sphere: $V = 4/3\pi r^3$ and $SA = 4\pi r^2$
- Square: $P = 4s$ and $A = s^2$
- Cube: $V = s^3$ and $SA = 6s^2$

Watch your units of measure and display the correct one, e.g., linear inches are different from square inches or cubic inches. *Think* to make sure you have got it right.

## 4 Code Specifications

- At the **bottom** of the program you should have a call to main().
- 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 is going on in the lines of code below, e.g., `# define circle functions`
- Use comments elsewhere as you think they help guide the reader. Do not 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.

## 5 Hints

- Use exactly the call hierarchy shown above, e.g., main calls displaySquareResults which calls squarePerim and squareArea. Do not “chain” functions to force execution order.
- Carefully consider parameters for each function. Think about what that function needs from outside (if anything) to do its work, or what it needs that it must pass on to functions it calls.
- One of the functions you will use returns more than one value; you will need to make use of Python’s ability to return multiple values.
- After you have getInput() working, start with one chain of calls and get them nicely fleshed out, e.g., main(), displaySquareResults() and squarePerim(). Then these can become templates for all other work (i.e., you can do lots of copying and pasting, but be aware of typical bugs this creates!).
- You must submit two .py files this time; zip them up and submit them as a single .zip file.

## 6 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 are not equipped to solve all problems you would like to; keep notes about what you would 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.

## 7 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 does not work as you would like, perhaps things that you would like to fix as you learn more?
- What did you learn from this assignment? What will you do differently on the next project?

## 8 Extra Credit

Create a PyUnit test script called TestGeometry.py. In it, create an automated test case for each of the functions in the geometry.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, find an "oracle" that calculates these independently, so you don't accidentally encode the same bad logic in both the functions and the tests.

## 9 Grading Matrix

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