In [ ]:
```
# Initialize Otter
import otter
grader = otter.Notebook("lesson_2.ipynb")
```

1 Lesson Two: Translating Physics into Python

Hi everyone! Welcome to SciTeens SOLID Physics Camp. Today you'll complete your second assignment. Please work through the problems below as best as you can. Feel free to message us if you have any difficulties.

What you should learn today: The importance of functions and how to use them, how to translate equations into Python, how to convert to different units using constants, and most importantly, what your weight would be on Jupiter.

1.1 Section One: Functions

Today we'll focus on the translation of physics ideas into Python. You'll easily get the hang of translating between the two, with some practice.

Mathematical operators look different in Python, you'll recall. For example, the quadratic $ x^2 - 4x + 4 $ can be written as the following cell in Python.

In [ ]:
```
x**2 - 4*x + 4
```

Note that the $ x^2 $ has to be written as x**2 in Python.

Also note that if you tried to run the above cell, you received an error. Try to read the error and understand what it says. The most important line is the one that says this: ***NameError: name 'x' not defined***. This tells us the type of error (Name) and what, specifically, we did wrong. In this case, we did not define the variable 'x'. Without defining a variable, the code will not work or do anything.

So, if we want a mathematical equation to function the same way we are used to, we have to put it in a function. Here, we've written the same quadratic as above, but in a function.

In [ ]:
```
def f(x):
    return x**2 - 4*x + 4
```

The input of the function is x, and the output is dictated by the return statement. Try running the cells below and note what they output. Is the notation similar to what you're used to seeing in math class?

```
In [ ]: f(0)
```

```
In [ ]: f(2)
```

```
In [ ]: f(-2)
```

Try running the following cell. Why doesn't it work the way the previous ones did?

```
In [ ]: f(2, 4)
```

What is the error trying to say? Look at the last line. It says it's a ***TypeError***, and that ***f() takes 1 positional argument but 2 were given***.

When we defined f(x), we wrote a function that only took 1 parameter: x. When we try to give it more than that, it doesn't know what to do. Similarly, when we don't give it any parameters, we get an error.

```
In [ ]: f()
```

Notice that we get another ***TypeError***, but the specifics are different. It says we're missing 1 positional argument, not that we have an extra one.

It's perfectly possible to have a function that takes no parameters or a function that takes more than 1 (there's no real upper limit), but it's important to stay true to the definition of the function. Because we defined f(x) as taking 1 parameter, we always have to give it 1 (exactly 1) when we call it.

Now that you've learned all the rules, you might be asking why functions are important. Instead of calling this:

```
In [ ]: f(400)
```

You could just do this, and it gives you the same result.

```
In [ ]: 400**2 - 4*400 + 4
```

In other words: why put something in a function if you could just write out exactly what you're doing and get the same result?

Learning Objective I: Why you should care about functions.

Sure, you can write everything out directly and not use functions, but it's a lot harder for a number of reasons.

Reason One: Readability Let's take the example we just used. Look at both cells and tell me what looks easier to read.

```
In [ ]: f(400)
```

```
In [ ]: 400**2 - 4*400 + 4
```

Using function notation is much more readable, especially in big blocks of code.

Reason Two: Efficiency Imagine you want to use the function you've defined, f(x), for every value in the range 0-100. What's easier, calling f(x) 100 times, or writing out the math 100 times?

Reason Three: It's Easier to Debug The term **debug** means, essentially, to fix errors (otherwise known as bugs) in code. Imagine if you had a really long chunk of code, with multiple functions being called.

If you isolate different tasks to different functions, it's easier to identify which parts are working correctly and which aren't. You can visualize this as a flowchart, like this:

If you don't separate parts of a problem out into functions, it's really difficult to discern where exactly you messed up.

1.1.1 Question One

Try writing the equation of a line $ y = 4x + 9 $ in Python by yourself

```
In [1]: def y(x):
            return 4 * x + 9 # SOLUTION
```

```
In [ ]: grader.check("q1")
```

1.1.2 Question Two

Try writing the equation of the polynomial $ g(x) = x^3 - 7x + 213 $. Test the following values: 0, -100, 12.

```
In [6]: def g(x):
            return x ** 3 - (7 * x) + 213 # SOLUTION
```

```
In [ ]: grader.check("q2")
```

1.1.3 Question Three

Write the equation of the polynomial r(x) = $ 6x^8 + 12x^7 - 9x^4 + 2x + 164$.

```
In [14]: def r(x):
             return (6 * x ** 8) + (12 * x ** 7) - (9 * x ** 4) + (2 * x) + 164 # SOLUTION
```

```
In [ ]: grader.check("q3")
```

Do you remember this equation from class? See what happens when you call the function.

Here, the variable v0 refers to initial velocity and x0 refers to initial position.

```
In [22]: def distance(acceleration, v0, x0, time):
             return(0.5*acceleration*(time*time) + v0*time + x0)
```

```
In [23]: distance(1, 1, 1, 1)
```

```
Out[23]: 2.5
```

Try calling the function using four of your own input values:

```
In [ ]: # your input
```

```
In [ ]: # your input
```

```
In [ ]: # your input
```

Here's another equation from class.

```
In [ ]: def velocity(v0, a, t):
            return(v0 + a*t)
```

Try calling this one.

```
In [ ]: # your input
```

```
In [ ]: # your input
```

```
In [24]: # your input
```

1.1.4 Question Four

Write the function 'kinematic' for our third equation from class below.

```
In [33]: def kinematic(v0, a, deltaX):
             return (v0) ** 2 + 2 * a * deltaX # SOLUTION
```

```
In [ ]: grader.check("q4")
```

What does your result represent? If your gut instinct is to say final velocity, you're close. Remember, this equation gives you the value of final velocity *squared*. If took the square root of our value, that'd be the value of final velocity.

1.2 Section Two: Constants and Conversions

In physics, we also have a lot of constants, such as acceleration due to gravity. In this section, we'll focus on constants, units of measurement, and conversion.

1.2.1 Conversions

For conversions, it is customary to write the variable name in all-caps. As with any variable, you can call it anything, but it helps other people who may be looking at your code to follow custom.

```
In [ ]: ACC_DUE_TO_GRAVITY = 9.8 # this initializes a constant
```

If, for example, we wanted to use the constant in a kinematic equation, we could just replace the acceleration variable. Here's an example.

```
In [ ]: def kinematic_2(v0, t):
            return(v0 + ACC_DUE_TO_GRAVITY*t)
```

Notice that we don't need to pass ACC_DUE_TO_GRAVITY as an argument because we've previously defined it.

Learning Objective II: Understanding the importance of conversions.

You might be wondering why conversions are important.

Hopefully, you recall a term called **SI** or **International System of Units**. It's important to use SI units in physics, because it allows your work to be easily understood by others.

Because everyone uses m/s/s as the SI unit for acceleration, they will know what you mean when you set a variable called ACC_DUE_TO_GRAVITY equal to 9.8.

1.2.2 Question Five

Recall from class that we can calculate the weight of an object by multiplying its mass by the acceleration due to gravity. How much does a 160 kilogram object weigh in Newtons at sea level?

```
In [35]: def kg_to_newtons(kg):
             return 9.8 * kg # SOLUTION
```

```
In [ ]: grader.check("q5")
```

For conversions, the same basic rules apply. Conventionally, conversion factors will also be written in all-caps and have very straightforward names. Here are a few.

```
In [ ]: MPH_TO_KPH = 1.6
        # conversion factor for miles per hour to kilometers per hour
```

```
In [ ]: FT_TO_M = 0.3
        # conversion factor for feet to meters
```

```
In [56]: S_TO_MIN = 60
         # conversion factor for seconds to minutes
```

Using the conversion factors given, complete the following:

1.2.3 Question Six

Convert 30 mph to kph, and assign it to the variable `kph`

```
In [48]: kph = 1.6 * 30 # SOLUTION
```

```
In [ ]: grader.check("q6")
```

1.2.4 Question Seven

How many seconds are in 2727 minutes? Assign your answer to the variable `seconds`

```
In [52]: seconds = 60 * 2727 # SOLUTION
```

```
In [ ]: grader.check("q7")
```

1.2.5 Question Eight

If an object has traveled a distance of 35 feet, how many meters has it covered? Assign this to the variable `meters`

```
In [57]: meters = 0.3 * 35 # SOLUTION
```

```
In [ ]: grader.check("q8")
```

These problems will require you to look up a conversion factor on your own. Assign the conversion factor to a variable (just like in the previous examples) and complete the problem.

1.2.6 Question Nine

If the weight of an object is 4899 metric tons, how much does it weigh in pounds? Assign this to the variable `pounds`

```
In [65]: pounds = 4899 * 2000 # SOLUTION
```

```
In [ ]: grader.check("q9")
```

1.2.7 Question Ten

If the volume of an object is 678 cubic centimeters, how much is that in cubic decimeters? Assign this to the variable `decimeters`

```
In [71]: decimeters = 678 / 1000 # SOLUTION
```

```
In [ ]: grader.check("q10")
```

1.3 Challenge Question

Calculate the weight of a 100-pound person in Newtons. Set this value to the variable `pounds_to_newtons`. As an extension, calculate the weight on Mars and Jupiter. The gravity on Mars is 3 m/s/s, and the gravity on Jupiter is 24.8 m/s/s. Set these values to the variables `mars` and `jupiter`, respectively. > Hint: Start with your weight in pounds, then move to kilograms, then Newtons. > This is a two-step problem.

```
In [51]: pounds_to_newtons = 9.8 * 0.45 * 100 # SOLUTION

         mars = 3 * 0.45 * 100 # SOLUTION

         jupiter = 24.8 * 0.45 * 100 # SOLUTION
```

In [ ]:
```
grader.check("challenge")
```

To double-check your work, the cell below will rerun all of the autograder tests.

In [ ]:
```
grader.check_all()
```

1.4 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

Please submit the resultant .zip file to the SciTeens platform

In [ ]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export(pdf=False)
```