Functions are ubiquitous in math classes, and unsurprisingly they are also helpful in programming. A function in math is a thing that has one or more inputs and has an output and the output is uniquely determined by the inputs.

In programming, functions can be created that are like mathematical functions (have inputs and an output determined by the inputs), or functions can just be chunks of code to run. In this class (and the entire sequence), I'll ask you to only write mathematical functions. For small programs like we're writing, the other uses of functions tend to have the effect of making your code less clear.

To create a function in Python (or in our pseudocode), you will write code like

```
def function_name(name_input1, name_input2):
    ... # a block of code that uses the inputs
    return EXPRESSION FOR VALUE
```

Note that the above is not actually working code. The first line defines the name of the function (replace `function_name` with a name that better describes what your function does), and it gives names to the **arguments**, which is to say the inputs. Those should also have descriptive names, which typically will match what you (or I) write in math notation. Definitely, do not name them `name_input1`!. There is a block of code indented (like you would with `if` or `while`), which contains in it at least one `return` statement, which specifies the value of the function, and also indicates that you are finished computing that value.

You can **call** a function by specifying its name, and then giving values for the arguments in parentheses. Something like:

```
value = function_name(5, 2.3)
```

Running code with functions on paper

```
def is_prime(n):
    for i in integers starting with 1 and ending with n-1:
        if n/i is an integer:
            return False
    return True

print(is_prime(3))
n = 4
if is_prime(2):
    print('That is what I thought!')
else:
    print('I think there is a bug.')
```

When running code that has a function in it, you read it from top to bottom, as usual, but when you see a `def`, you basically skip over that block and write down a variable value that (in theory) contains the entire function definition. Then when a function is called (which you recognize by a variable beging followed by a parenthesis (), you compute the values provided as inputs, and then jump into the definition of the function (remembering where you came from) and assign the values computed to the argument names. Then you run the function block until you hit a return. When you hit a return, you evaluate the return expression, and then jump back to where you came from, and use that value (which was `return`ed) for the value of the function call expression.

Your task: factorial

1. Write a function to compute the factorial $n!$, which is defined as the product of all integers up to and including n.
2. Test your function by computing and printing the factorials of integers from 0 to 5.
3. Just for fun, compute 52!, which is the number of ways a deck of cards can be shuffled.

Your task: approximating a cosine A Taylor series can be used to create an approximation for a function, which we call a *power series approximation*. The Taylor series for cosine is:

$$\cos(\theta) = \sum_{i=0}^{\text{even}} \frac{(-1)^{i/2}}{i!}\theta^i \tag{1}$$

where the sum goes to ∞ over even values of i.

1. Write a function that given θ and i_{max} computes the Taylor series of $\cos\theta$ including the terms up to and including $i = i_{\text{max}}$.
2. Test your function by printing cos(0.01) and a few approximations of this with moderate i_{max}.
3. Plot $\cos\theta$ and be sure to label your axes.
4. Plot your Taylor series approximation for i_{max} of 0, 1, 2, and 3. Use a legend to distinguish between your curves.

Exra fun Solve for the Taylor series for e^x (expanded around $x = 0$), and write a program to solve for an approximation for the exponential including terms up to n. Plot and compare the exponential with your approximation.

$\sqrt{\textbf{fun}}$ Try using a Taylor series approximation on $\frac{1}{\sqrt{1+z}}$.

Error fun Plot the error in the cosine approximation, as a function of θ. Or at fixed θ, plot the error as a function of i_{max}.