In your pairs, please run the following code by hand on your big whiteboards (or paper), to predict what it will print

```
i = 1
def factorial(n):
    i = n
    product = 1
    while i > 0:
        product = product * i
        i = i - 1
    print('inside factorial i is', i)
    return product
print('i starts as', i)
print('factorial is', factorial(3))
print('i ends as', i)
print('product is', product)
```

The above program will actually print:

```
i starts as 1
inside factorial i is 0
factorial is 6
i ends as 1
error message: product is undefined
```

Many of you may have expected it to print `i is 0` both of the last two times. This difference relates to the **scoping** rules for variables. In this program there are actually *two* variables named `i`. One is the **global variable** whose value is `1` throughout the program. The other is a **local variable** that is created when the function is called, whose value is changed during the computation of the factorial.

When a function is called, python creates new local variables for each variable that is assigned to in the function. When the function returns, all of these variables are erased. This may feel like weird and unintuitive behavior, but it is crucial for allowing larger programs to be read successfully. In particular, it means:

1. You can understand a function by reading just that function in isolation, provided it doesn't use any variables that are neither input parameters nor initialized within the function itself.

2. If you know what a function *does*, you don't need to worry about how it does it. You don't need to be concerned that you might accidentally use the same variable name that is unsed inside of some function and get messed up because your variable got modified by the function. ***This is huge!***

3. By the same token, when writing a function, you don't need to worry whether some variable name might be used elsewhere in the code. ***This is huge!***

Let's run as a class

```
x = 1
hi = 'Greetings fellow humans!'
def sqrt(x):
    print('inside sqrt x is', x)
    lo = 0
    hi = x
    while hi - lo > 0.2:
        print(lo, '< root x <', hi)
        mid = (lo + hi)/2
        if mid*mid > x:
            hi = mid
        else:
            lo = mid
    return (hi + lo)/2
print(hi)
print('x starts as', x)
print('sqrt(2) is', sqrt(2))
print('x ends as', x)
print(hi)
```

Try running by hand

```
x = 1
def sqrt(x):
    print('inside sqrt x is', x)
    return x**0.5
print('x starts as', x)
print('sqrt(5) is', sqrt(5))
print('x ends as', x)
```

Try running by hand

```
i = 1
def factorial(n):
    product = 1
    while i <= n:
        product = product * i
        i = i + 1
    print('inside i is', i)
    return product
print('i is', i)
print('3! is', factorial(3))
print('i is', i)
```

Then try running using the computer, copying this code into an entirely new file.

Try running by hand

```
i = 1
def factorial(n):
    product = 1
    for i in range(1, n+1):
        product = product * i
    print('inside i is', i)
    return product
print('i is', i)
print('3! is', factorial(3))
print('i is', i)
```

Then try running using the computer, copying this code into an entirely new file.