Functions:

*** Personal Notes also included here. In-clas work below! ***

*args:

In [3]:
```
def arg_expansion_example(x, y):
    return x**y

my_args = [2, 8]
arg_expansion_example(*my_args) # telling it we do not know how many parameters
```

Out[3]: 256

Default Arguments:

In [4]:
```
def show_results(precision, printing=True):
    precision = round(precision, 2)
    if printing:
      print('precision =', precision)
    return precision

pr = 0.912
res = show_results(pr)
```

```
precision = 0.91
```

In [8]:
```
x = show_results(pr, False) # now it turns the printing off.
```

In [9]:
```
show_results(pr, False)
```

Out[9]: 0.91

Returning Values:

In [10]:
```
## returns three values

def negate_coords(x, y, z):
    return -x, -y, -z

a, b, c = negate_coords(10, 20, 30)
print('a =', a)
print('b =', b)
print('c =', c)
```

```
a = -10
b = -20
c = -30
```

In [12]:
```
foo = negate_coords(10, 20, 30) # returns a Tuple!
foo, len(foo)
```

Out[12]: ((-10, -20, -30), 3)

In [14]:
```
def absolute_value(num):
    if num >= 0:
        return num
    return -num

absolute_value(-4)

# For non-negative values, the first return is reached.
# For negative values, the second return is reached.
```

Out[14]: 4

Lambda Functions:

Python lambda functions are small, informal functions. They don't get a name.

The are "anonymous" or "unnamed".

In [21]:
```
my_lambda = lambda x: x + 1

result = my_lambda(5)
print(result)

my_lambda(8)
```

6

Out[21]: 9

In [2]:
```
# Package first element and all data into tuple

pack_first_all = lambda x: (x[0], x)

casado = ('rice','beans','salad','plaintain','chicken') # a typical Costa Rican

pack_first_all(casado)
```

Out[2]: ('rice', ('rice', 'beans', 'salad', 'plaintain', 'chicken'))

In [3]:
```
# check for keyword "dirty"

is_dirty = lambda txt: 'dirty' in txt

kitchen_inspection = 'dirty dishes'
is_dirty(kitchen_inspection)
```

Out[3]: True

Recursion:

A recursive function is a **function that calls itself.**

recursion - the art of defining something (at least partly) in terms of itself, which is a naughty no-no in dictionaries but often works out okay in computer programs if you're careful not to recurse forever (which is like an infinite loop with more spectacular failure modes).

Should try to stay away from recursion!

In [8]:
```
n = 5

def factorial_for(x):
    "Finds the factorial of an integer using a for loop"
    f = x
    for i in range(1, x):
        x -= 1
        f *= x
    return f

%time factorial_for(n) # able to see the run-time.
```

```
CPU times: total: 0 ns
Wall time: 0 ns
```

Out[8]:
```
120
```

M04 Exercises

In [10]:
```
import numpy as np
import pandas as pd
```

4.1.

In [19]:
```
def length_string(str, var):
    if len(str) == len(var):
        print("True")
    else:
        print("False")
    print(len(var))

# Example usage
length_string("is everything okay?", "hi")
```

```
False
2
```

4.2.

In [21]:
```
def square_args(*vars):
    for var in vars:
        print(var**2)
    return None

square_args(2)
square_args(10, 2, 8)
```

```
4
100
4
64
```

4.3.

In [24]:
```
shorter = lambda x: f"{x[0]}{len(x[1:-1])}{x[-1]}".upper()
shorter("operationalization")
```

Out[24]:
```
'O16N'
```

4.4.

In [30]:
```
def num_check(number):
    number -= 5
    if number < 0:
        return 1
    return number

num_check(4)
```

Out[30]:
```
1
```

In [ ]:
```

```