We already learned to create functions which accept a parameter and return values

```
def get_initial(name):
      initial = name[0:1].upper()
      return initial

first_name = input('Enter your first name: ')
first_name_initial = get_initial(first_name)

print('Your initial is: ' + first_name_initial)
```

```
# output
Enter your first name: adam
Your initial is: A
```

Functions can accept multiple parameters

```
def get_initial(name, force_uppercase):
    if force_uppercase:
        initial = name[0:1].upper()
    else:
        initial = name[0:1]
    return initial

first_name = input('Enter your first name: ')
first_name_initial = get_initial(first_name, False)

print('Your initial is: ' + first_name_initial)
```

Pass the parameters in the same order they are listed in the function declaration

```
# output
Enter your first name: adam
Your initial is: a
```

You can specify a default value for a parameter

```
def get_initial(name, force_uppercase=True):
    if force_uppercase:
        initial = name[0:1].upper()
    else:
        initial = name[0:1]
    return initial

first_name = input('Enter your first name: ')
first_name_initial = get_initial(first_name)

print('Your initial is: ' + first_name_initial)
```

```
# output
Enter your first name: adam
Your initial is: A
```

You can also assign the values to parameters by name when you call the function

```
def get_initial(name, force_uppercase):
    if force_uppercase:
        initial = name[0:1].upper()
    else:
        initial = name[0:1]
     return initial

first_name = input('Enter your first name: ')
first_name_initial = get_initial(force_uppercase=True,\
                                 name=first_name)
```

When you use named parameters, you can specify parameters in any order

```
# output
Enter your first name: adam
Your initial is: A
```

Using the named notation when calling functions makes your code more readable

```
def error_logger(error_code, error_severity, log_to_db, \
                 error_message, source_module):
    print('oh no error: ' + error_message)
    # Imagine code here that logs our error to a database or file

first_number = 10
second_number = 5
if first_number > second_number:
    error_logger(45, 1, True,
                 'Second number greater than first',
                 'my_math_method')
```

Using the named notation when calling functions makes your code more readable

```
def error_logger(error_code, error_severity, log_to_db, \
                 error_message, source_module):
    print('oh no error: ' + error_message)
    # Imagine code here that logs our error to a database or file

first_number = 10
second_number = 5
if first_number > second_number:
    error_logger(error_code=45, error_severity=1,
                 log_to_db=True,
                 error_message='Second number greater than first',
                 source_module='my_math_method')
```