Functions

Defining a Function

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python greeter_
1.py
Hello!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Passing Information to a Function

```
def greet_user(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user('jesse')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python greeter_
2.py
Hello, Jesse!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Arguments and Parameters

- The variable `username` in the definition of `greet_user()` is an example of a *parameter*, a piece of information the function needs to do its job.
- The value `'jesse'` in `greet_user('jesse')` is an example of an *argument*.

Passing Arguments

Positional Arguments

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_1.p
y

I have a hamster.
My hamster's name is Harry.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Multiple Function Calls

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
describe_pet('dog', 'willie')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_2.p
y

I have a hamster.
My hamster's name is Harry.

I have a dog.
My dog's name is Willie.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Order Matters in Positional Arguments

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('harry', 'hamster')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_3.p
y

I have a harry.
My harry's name is Hamster.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Keyword Arguments

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_4.p
y

I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Default Values

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(pet_name='willie')
describe_pet('jimmy')
describe_pet(pet_name='harry', animal_type='hamster')
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets.py

I have a dog.
My dog's name is Willie.

I have a dog.
My dog's name is Jimmy.

I have a hamster.
My hamster's name is Harry.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Equivalent Function Calls

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

# A dog named Willie.
describe_pet('willie')
describe_pet(pet_name='willie')
# A hamster named Harry.
describe_pet('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_5.p
y

I have a dog.
My dog's name is Willie.

I have a dog.
My dog's name is Willie.

I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Avoiding Argument Errors

```
def describe_pet(pet_name, animal_type):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet()
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ python pets_6.p
y
Traceback (most recent call last):
  File "pets_6.py", line 6, in <module>
    describe_pet()
TypeError: describe_pet() missing 2 required positional arguments: 'pet_name' and 'animal_type'
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Return Values

Returning a Simple Value

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
~
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on formatted_name_1.py
Jimi Hendrix
```

Making an Argument Optional

```
def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on formatted_name.py
Jimi Hendrix
John Lee Hooker
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Returning a Dictionary

```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on person.py
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Using a Function with a `while` Loop

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

# This is an infinite loop!
while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")

    f_name = input("First name: ")
    if f_name == 'q':
        break

    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on greeter.py

Please tell me your name:
(enter 'q' at any time to quit)
First name: john
Last name: smith

Hello, John Smith!

Please tell me your name:
(enter 'q' at any time to quit)
First name: q
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Passing a List

```
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = f"Hello, {name.title()}!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on greet_users.py
Hello, Hannah!
Hello, Ty!
Hello, Margot!
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Modifying a List in a Function

```
def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until none are left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)

unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []

print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on printing_models.py
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Preventing a Function from Modifying a List

- You can address this issue by passing the function a copy of the list, not the original.
- Any changes the function makes to the list will affect only the copy, leaving the original list intact.
- You can send a copy of a list to a function like this:
  `function_name(list_name[:])`

```python
def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until none are left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)

unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []

print_models(unprinted_designs[:], completed_models)
show_completed_models(completed_models)
print(unprinted_designs)
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on printing_models_1.py
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case
['phone case', 'robot pendant', 'dodecahedron']
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Passing an Arbitrary Number of Arguments

```
def make_pizza(*toppings):
    """Print the list of toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on pizza_1.py
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
def make_pizza(*toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on pizza_2.py

Making a pizza with the following toppings:
- pepperoni

Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Mixing Positional and Arbitrary Arguments

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on pizza_3.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Using Arbitrary Keyword Arguments

```
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)
~
~
~
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on user_profile.py
{'location': 'princeton', 'field': 'physics', 'first_name': 'albert', 'last_name': 'einstein'}
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Storing Your Functions in Modules

- You can store your functions in a separate file called a *module* and then *importing* that module into your main program.
- An `import` statement tells Python to make the code in a module available in the currently running program file.

Importing an Entire Module

- A *module* is a file ending in *.py* that contains the code you want to import into your program.

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
pizza.py
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
making_pizzas.py
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on making_pizzas.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Importing Specific Functions

- You can also import a specific function from a module.
- Here's the general syntax for this approach:
  `from module_name import function_name`
- You can import as many functions as you want from a module by separating each function's name with a comma:
  `from module_name import function_0, function_1, function_2`

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
making_pizzas_1.py
from pizza import make_pizza

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on making_pizzas_1.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Using as to Give a Function an Alias

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
making_pizzas_2.py
from pizza import make_pizza as mp

mp(16, 'pepperoni')
mp(12, 'mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on making_pizzas_2.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Using as to Give a Module an Alias

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
pizza_4.py
import pizza as p

p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on pizza_4.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Importing All Functions in a Module

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ cat
pizza_5.py
from pizza import *

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

```
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$ pyth
on pizza_5.py

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
(base) joshua@joshua-VirtualBox:~/Documents/Python_Crash_Course_2nd_Edition/ehmatthes-pcc_2e-00ff4d9/chapter_08$
```

Styling Functions

- You need to keep a few details in mind when you’re styling functions.
- Functions should have descriptive names, and these names should use lowercase letters and underscores.
- Descriptive names help you and others understand what your code is trying to do.
- Module names should use these conventions as well.

- Every function should have a comment that explains concisely what the function does.
- This comment should appear immediately after the function definition and use the docstring format.
- In a well-documented function, other programmers can use the function by reading only the description in the docstring.
- They should be able to trust that the code works as described, and as long as they know the name of the function, the arguments it needs, and the kind of value it returns, they should be able to use it in their programs.

- If you specify a default value for a parameter, no spaces should be used on either side of the equal sign:
  ```
  def function_name(parameter_0, parameter_1='default value')
  ```
- The same convention should be used for keyword arguments in function calls:
  ```
  function_name(value_0, parameter_1='value')
  ```

- PEP 8 (https://www.python.org/dev/peps/pep-0008/) recommends that you limit lines of code to 79 characters so every line is visible in a reasonably sized editor window.
- If a set of parameters causes a function's definition to be longer than 79 characters, press `ENTER` after the opening parenthesis on the definition line.
- On the next line, press `TAB` twice to separate the list of arguments from the body of the function, which will only be indented one level.

- def *function_name*(
        parameter_0, *parameter_1*, *parameter_2*,
        parameter_3, *parameter_4*, *parameter_5*):
    function body...

- If your program or module has more than one function, you can separate each by two blank lines to make it easier to see where one function ends and the next one begins.

- All `import` statements should be written at the beginning of a file.
- The only exception is if you use comments at the beginning of your file to describe the overall program.