Functions (Chapter 8)

Functions are named blocks of code that are designed to do one specific job. Functions are useful to:

- *Reduce the amount of code you need to write*- the same function can be called as many times as needed
- *Make your code more readable*- breaking code into smaller blocks with meaningful names and roles
- *Make it easier to resuse code*- if you need the same function again in the future just `import` it

In [1]:
```
# defining a function
def greet_user():
    """Display a simple greeting"""
    print("Hello!")

greet_user()
print("\n")
help(greet_user) # built in help function prints the comment below the function name
```

```
Hello!


Help on function greet_user in module __main__:

greet_user()
    Display a simple greeting
```

In [2]:
```
# passing in information (parameters)
def greet_user(username):
    """Display a simple greeting"""
    print(f"Hello, {username.title()}!")
```

```
greet_user("alice")
```

```
Hello, Alice!
```

In [3]:
```
# 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')
# order matters!
describe_pet('harry','hamster')
```

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

I have a harry.
My harry's name is Hamster.
```

In [4]:
```
# keyword arguments
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(pet_name='willie', animal_type='whale')
# order does not matter
describe_pet(animal_type='whale',pet_name='willie')
```

```
I have a whale.
My whale's name is Willie.

I have a whale.
My whale's name is Willie.
```

In [5]:
```
# 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('davis')
```

```
I have a dog.
My dog's name is Davis.
```

In [6]:
```python
# equivalent function calls

# keyword
describe_pet(pet_name='davis')
# positional
describe_pet('davis')
# combination (using same value as default)
describe_pet('davis',animal_type='dog')
```

```
I have a dog.
My dog's name is Davis.

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

I have a dog.
My dog's name is Davis.
```

Return Values

Functions can optionally return a result rather than outputing data directly.

In [10]:
```python
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)
#get_formatted_name('jimi', 'hendrix')
```

```
Jimi Hendrix
```

In [11]:
```python
# optional arguments
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)
```

```
Jimi Hendrix
John Lee Hooker
```

In [12]:
```
# returning other data types like 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)
```

```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
```

In [1]:
```
# using functions 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}!")
```

```
Please tell me your name:
(enter 'q' at any time to quit)
First name: Erin
Last name: Kincade

Hello, Erin Kincade!

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

Passing Lists as Parameters

In [2]:
```python
# 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)
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

In [3]:
```python
# 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)
print(f"\nThere are {len(unprinted_designs)} models left to be printed")
```

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case

There are 0 models left to be printed
```

In [4]:
```
# preventing a function from modifying a list
# pass in a copy using the slice notation

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

print_models(unprinted_designs[:], completed_models) # <--- slice notation for unprinted_designs
show_completed_models(completed_models)
print(f"\nThere are {len(unprinted_designs)} models left to be printed")
```

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: phone case

The following models have been printed:
dodecahedron
robot pendant
phone case

There are 3 models left to be printed
```

Passing an Arbitrary Number of Arguments

Useful if you do not know how many arguments you will need ahead of time. In practice this is not very common. Try to avoid using in your code but you may find it in libraries that you use in the future.

In [5]:
```
def make_pizza(*toppings):
    print(toppings)

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

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

In [6]:
```
# add a loop to produce formatted output

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

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

```
Making a pizza with the following toppings:
- pepperoni

Making a pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

In [7]:
```
# 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')
```

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

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Storing Functions in Modules

Modules are useful for separating large programs into multiple files with groups of similar functions. This also makes it easier to reuse your code in the future.

In [8]:
```python
# importing an entire module (recommended!)
import pizza

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[8], line 2
      1 # importing an entire module (recommended!)
----> 2 import pizza
      4 pizza.make_pizza(16, 'pepperoni')
      5 pizza.make_pizza(12, 'mushrooms','green peppers','extra cheese')

ModuleNotFoundError: No module named 'pizza'
```

In [9]:
```python
# importing a specific function (less recommended)
from pizza import make_pizza

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[9], line 2
      1 # importing a specific function (less recommended)
----> 2 from pizza import make_pizza
      4 make_pizza(16, 'pepperoni')
      5 make_pizza(12, 'mushrooms','green peppers','extra cheese')

ModuleNotFoundError: No module named 'pizza'
```

In [10]:

```
# using aliases (try to avoid)
from pizza import make_pizza as mp
mp(16, 'pepperoni')
mp(12, 'mushrooms','green peppers','extra cheese')

import pizza as p
p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms','green peppers','extra cheese')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[10], line 2
      1 # using aliases (try to avoid)
----> 2 from pizza import make_pizza as mp
      3 mp(16, 'pepperoni')
      4 mp(12, 'mushrooms','green peppers','extra cheese')

ModuleNotFoundError: No module named 'pizza'
```

In [11]:

```
# importing all functions in a module (really try to avoid!)

def say_hello():
    print("\nHello world!")

from pizza import *
make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms','green peppers','extra cheese')

say_hello() # runs the pizza module's function!
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[11], line 6
      3 def say_hello():
      4     print("\nHello world!")
----> 6 from pizza import *
      7 make_pizza(16, 'pepperoni')
      8 make_pizza(12, 'mushrooms','green peppers','extra cheese')

ModuleNotFoundError: No module named 'pizza'
```

Homework Problems

8-1. Message: Write a function called `display_message()` that prints one sentence telling everyone what you are learning about in this chapter. Call the function, and make sure the message displays correctly.

In [12]:
```
def display_message():
    print('I am learning about functions!')

display_message()
```

```
I am learning about functions!
```

8-2. Favorite Book: Write a function called `favorite_book()` that accepts one parameter, `title`. The function should print a message, such as *One of my favorite books is Alice in Wonderland*. Call the function, making sure to include a book title as an argument in the function call.

In [18]:
```
def favorite_book(title):
    print(f'One of my favorite books is {title}')

favorite_book('Alice in Wonderland')
```

```
One of my favorite books is Alice in Wonderland
```

8-3. T-Shirt: Write a function called `make_shirt()` that accepts a size and the text of a message that should be printed on the shirt. The function should print a sentence summarizing the size of the shirt and the message printed on it.

Call the function using positional arguments to make a shirt. Call the function a second time using keyword arguments.

In [19]:
```
def make_shirt(size,text):
    print(f'your shirt will be {size} with {text} displayed on it')

s = input('size?: ')
words = input('text?: ')
make_shirt(s,words)
```

```
size?: Large
text?: hi.
your shirt will be Large with hi. displayed on it
```

8-4. Large Shirts: Modify the `make_shirt()` function so that shirts are large by default with a message that reads *I love Python*. Make a large shirt and a medium shirt with the default message, and a shirt of any size with a different message.

In [21]:
```
def make_shirt(size = 'Large',text = 'I love python'):
    print(f'your shirt will be {size} with {text} displayed on it')

#s = input('size other than Large?: ')
#if s != 'Large':
 #   size = s
#words = input('alternate text?: ')
#if words != 'I love python':
 #   text = words
make_shirt()
make_shirt('medium')
make_shirt('small', 'hi')
```

```
your shirt will be Large with I love python displayed on it
your shirt will be medium with I love python displayed on it
your shirt will be small with hi displayed on it
```

8-5. Cities: Write a function called `describe_city()` that accepts the name of a city and its country. The function should print a simple sentence, such as *Reykjavik is in Iceland*. Give the parameter for the country a default value. Call your function for three different cities, at least one of which is not in the default country (pass the correct country as a keyword parameter).

In [23]:
```
def describe_city(city, country='USA'):
    print(f'{city} is in {country}')

describe_city('ATL')
describe_city('NYC')
describe_city('Budapest', 'Hungary')
```

```
ATL is in USA
NYC is in USA
Budapest is in Hungary
```

8-6. City Names: Write a function called `city_country()` that takes in the name of a city and its country. The function should return a string formatted like this:

```
"Santiago, Chile"
```

Call your function with at least three city-country pairs, and print the values that are returned.

In [25]:
```
def city_country(city,country):
    print(f'{city.title()}, {country.title()}')

city_country("ATL","USA")
city_country("NYC","USA")
city_country("Budapest","Hungary")
```

```
Atl, Usa
Nyc, Usa
Budapest, Hungary
```

8-7. Album: Write a function called `make_album()` that builds a dictionary describing a music album. The function should take in an artist name and an album title, and it should return a dictionary containing these two pieces of information. Use the function to make three dictionaries representing different albums. Print each return value to show that the dictionaries are storing the album information correctly.

Use `None` to add an optional parameter to `make_album()` that alows you to store the number of songs on an album. If the calling line includes a values for the number of songs, add that value to the album's dictionary. Make at least one new function call that includes the number of songs on an album.

In [41]:
```
def make_album(singer, album_title, songs='unknown'):
    album = {'artist': singer,
            'album': album_title,
            'num_songs': songs}
    print(album)
make_album('Taylor', '1989')
make_album('Drake', 'For All the Dogs')
make_album('Post Malone', "AUSTIN")
make_album('Taylor', 'Red', '10')
```

```
{'artist': 'Taylor', 'album': '1989', 'num_songs': 'unknown'}
{'artist': 'Drake', 'album': 'For All the Dogs', 'num_songs': 'unknown'}
{'artist': 'Post Malone', 'album': 'AUSTIN', 'num_songs': 'unknown'}
{'artist': 'Taylor', 'album': 'Red', 'num_songs': '10'}
```

8-8. User Albums: Start with your program from Exercise 8-7. Write a `while` loop that allows users to enter an album's artist and title. Once you have that information, call `make_album()` with the users' input and print the dictionary that's created. Be sure to include a quit value in the while loop.

In [47]:
```
c = 'please run'
while c != 'n':
    c = input('would you like to continue (y/n)?: ')
    if c == 'y':
        artist = input('artist name: ')
        album_title = input('album title: ')
        make_album(artist, album_title)
```

```
would you like to continue (y/n)?: y
artist name: taylor
album title: 1989
{'artist': 'taylor', 'album': '1989', 'num_songs': 'unknown'}
would you like to continue (y/n)?: n
```

8-9. Messages: Make a list containing a series of short text messages. Pass the list to a function called `show_messages()`, which prints each text message.

In [48]:
```
def show_messages(messages):
    for message in messages:
        print(message)


messages = ['hi', 'yes', 'no', 'how are you?']
show_messages(messages)
```

```
hi
yes
no
how are you?
```

8-10. Sending Messages: Start with a copy of your program from Exercise 8-9. Write a function called `send_messages()` that prints each text message and moves each message to a new list called `sent_messages` as it's printed. After calling the function, print both of your lists to make sure the messages were moved correctly.

In [55]:
```
sent_messages = []
def send_messages(messages):
    for message in messages:
        print(message)
        sent_messages.append(message)
       # messages.remove(message)
    for message in sent_messages:
        messages.remove(message)




messages = ['hi', 'yes', 'no', 'how are you?']
send_messages(messages)
print(messages)
print(sent_messages)
```

```
hi
yes
no
how are you?
[]
['hi', 'yes', 'no', 'how are you?']
```

8-11. Archived Messages: Start with your work from Exercise 8-10. Call the function `send_messages()` with a copy of the list of messages. After calling the function, print both of your lists to show that the original list has retained its messages.

In [56]:
```
sent_messages = []
def send_messages(messages):
    for message in messages:
        print(message)
        sent_messages.append(message)
       # messages.remove(message)
    #for message in sent_messages:
       # messages.remove(message)




messages = ['hi', 'yes', 'no', 'how are you?']
send_messages(messages)
print(messages)
print(sent_messages)
```

```
hi
yes
no
how are you?
['hi', 'yes', 'no', 'how are you?']
['hi', 'yes', 'no', 'how are you?']
```

8-12. Sandwiches: Write a function that accepts a list of items a person wants on a sandwich. The function should have one parameter that collects as many items as the function call provides, and it should print a summary of the sandwich that's being ordered. Call the function three times, using a different number of arguments each time.

In [57]:
```
def sammy(items):
    print(f'your sandwich will have {items} on it')

toppings = input('what would you like on your sandwich?')
sammy(toppings)
```

```
what would you like on your sandwich?ham, cheese, lettuce, tomato
your sandwich will have ham, cheese, lettuce, tomato on it
```

8-13. User Profile: Start with a copy of the user_profile code above (or on page 149). Build a profile of yourself by calling `build_profile()`, using yourfirst and last names and three other key-value pairs that describe you.

In [1]:
```
def build_profile(first, last, hair, height, state):
    print(f'{first.title()} {last.title()} has {hair} hair is {height} tall and is from {state}.')

build_profile('erin','kincade','brown', "5'7", 'GA')
```

```
Erin Kincade has brown hair is 5'7 tall and is from GA.
```

8-14. Cars: Write a function that stores information about a car in a dictionary. The function should always receive a manufacturer and a model name. It should then accept an arbitrary number of keyword arguments. Call the function with the required information and two other name-value pairs, such as a color or an optional feature. Your function should work for a call like this one:

```
car = make_car('subaru', 'outback', color='blue', tow_package=True)
```

Print the dictionary that's returned to make sure all the information was stored correctly.

In [5]:
```
def car(manu,model,color,doors):
    cars={}
```

```
    cars['manufacturer'] = manu
    cars['model'] = model
    cars['color'] = color
    cars['doors'] = doors
    print(cars)

car('subaru','legacy','silver','4')
```

```
{'manufacturer': 'subaru', 'model': 'legacy', 'color': 'silver', 'doors': '4'}
```

8-15. Printing Models: Move the functions below into a separate file called *printing_functions.py* Write an `import` statement at the top of the code below so the code uses the module's functions.

In [9]:
```
#TODO: Move to printing_functions.py
import printing_functions

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[9], line 2
      1 #TODO: Move to printing_functions.py
----> 2 import printing_functions
      4 print_models(unprinted_designs, completed_models)
      5 show_completed_models(completed_models)

ModuleNotFoundError: No module named 'printing_functions'
```

8-16.Imports: Create a file called hello.py with a function that prints a greeting. Import the function using each of these approaches and verify it works by calling the function (you should see 6 copies of the greeting when you run the code)

In [10]:
```
import hello
hello('hi')
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
Cell In[10], line 1
----> 1 import hello
      2 hello('hi')

ModuleNotFoundError: No module named 'hello'
```

In [ ]: