Task 7

Exercise 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 [ ]:
```
def displayMessage():
    print("I am learning to use the function!")
    print("This is fun!")

displayMessage()
```

```
I am learning to use the function!
This is fun!
```

Exercise 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 [ ]:
```
def favoriteBook(title):
    print("One of my favorite books is " + title)

favoriteBook("Kafka on the Shore")
```

```
One of my favorite books is Kafka on the Shore
```

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

In [ ]:
```
def makeShirt(size, message):
    print("The shirt will be a " + size + " and will say " + message)

makeShirt("large", "I ♥ Bytewise")
makeShirt(message="I ♥ Bytewise", size="large")
```

```
The shirt will be a large and will say I ♥ Bytewise
The shirt will be a large and will say I ♥ Bytewise
```

Exercise 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 [ ]:
```
def makeShirt(size="large", message="I love Python"):
    print("The shirt will be a " + size + " and will say " + message)

makeShirt()
makeShirt(size="medium")
makeShirt("small", "Programmers are cool!")
```

```
The shirt will be a large and will say I love Python
The shirt will be a medium and will say I love Python
The shirt will be a small and will say Programmers are cool!
```

Exercise 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.

In [ ]:
```
def describeCity(city, country="USA"):
    print(city + " is in " + country)

describeCity("New York")
describeCity("Paris", "France")
describeCity("Tokyo", "Japan")
```

```
New York is in USA
Paris is in France
Tokyo is in Japan
```

Exercise 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 value that's returned.

In [ ]:
```
def cityCountry(city, country):
    return f"{city.title()}, {country.title()}"

print(cityCountry("new york", "usa"))
print(cityCountry("paris", "france"))
print(cityCountry("tokyo", "japan"))
```

```
New York, Usa
Paris, France
Tokyo, Japan
```

Exercise 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.
Add an optional parameter to `make_album()` that allows you to store the number of tracks on an album. If the calling line includes a value for the number of tracks, add that value to the album's dictionary. Make at least one new function call that includes the number of tracks on an album.

In [ ]:
```
def makeAlbum(artist, title):
    return {"artist": artist, "title": title}

album=makeAlbum("The Beatles", "Abbey Road")
print(album)

album=makeAlbum("The Weeknd", "Starboy")
print(album)

album=makeAlbum("The Weeknd", "Beauty Behind the Madness")
print(album)
```

```
{'artist': 'The Beatles', 'title': 'Abbey Road'}
{'artist': 'The Weeknd', 'title': 'Starboy'}
{'artist': 'The Weeknd', 'title': 'Beauty Behind the Madness'}
```

In [ ]:
```
def makeAlbum(artist, title, tracks=None):
    album = {"artist": artist, "title": title}
    if tracks:
        album["tracks"] = tracks
    return album

album=makeAlbum("The Beatles", "Abbey Road", 17)
print(album)

album=makeAlbum("The Weeknd", "Starboy", 18)
print(album)

album=makeAlbum("The Weeknd", "Beauty Behind the Madness", 18)
print(album)
```

```
{'artist': 'The Beatles', 'title': 'Abbey Road', 'tracks': 17}
{'artist': 'The Weeknd', 'title': 'Starboy', 'tracks': 18}
{'artist': 'The Weeknd', 'title': 'Beauty Behind the Madness', 'tracks': 18}
```

Exercise 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 user's input and print the dictionary that's created. Be sure to include a quit value in the while loop.

In [ ]:
```
def makeAlbum(artist, title, tracks=None):
    album = {"artist": artist, "title": title}
    if tracks:
        album["tracks"] = tracks
    return album

print("Enter 'q' at any time to quit.")

while True:
    title=input("Enter the title of the album: ")
    if title == "q":
        break
    artist=input("Enter the artist of the album: ")
    if artist == "q":
        break
    album=makeAlbum(artist, title)
    print(album)

print("\nThanks for using the program!")
```

```
Enter 'q' at any time to quit.
{'artist': 'The Weeknd', 'title': 'Starboy'}
{'artist': 'Kendrick Lamar', 'title': 'DAMN'}
{'artist': 'Joji', 'title': 'Smithereens'}

Thanks for using the program!
```

Exercise 8.9: Magicians.

Make a list of magician's names. Pass the list to a function called `show_magicians()`, which prints the name of each magician in the list.

In [ ]:
```
def showMagicians(magicians):
    for magician in magicians:
        print(magician)

magicians=["David Copperfield", "David Blaine", "Criss Angel"]
showMagicians(magicians)
```

```
David Copperfield
David Blaine
Criss Angel
```

Exercise 8.10: Great Magicians.

Start with a copy of your program from Exercise 8.9. Write a function called `make_great()` that modifies the list of magicians by adding the phrase the Great to each magician's name. Call `show_magicians()` to see that the list has actually been modified.

In [ ]:
```
def showMagicians(magicians):
    for magician in magicians:
        print(magician)
```

```
def makeGreat(magicians):
    greatMagicians=[]
    while magicians:
        magician=magicians.pop()
        greatMagician=magician + " the Great"
        greatMagicians.append(greatMagician)
    for greatMagician in greatMagicians:
        magicians.append(greatMagician)

magicians=["David Copperfield", "David Blaine", "Criss Angel"]
makeGreat(magicians)
showMagicians(magicians)
```

```
Criss Angel the Great
David Blaine the Great
David Copperfield the Great
```

Exercise 8.11: Unchanged Magicians.

Start with your work from Exercise 8.10. Call the function `make_great()` with a copy of the list of magicians' names. Because the original list will be unchanged, return the new list and store it in a separate list. Call `show_magicians()` with each list to show that you have one list of the original names and one list with the Great added to each magician's name.

In [ ]:
```
def showMagicians(magicians, greatMagicians):
    for magician in magicians:
        print(magician)
    for greatMagician in greatMagicians:
        print(greatMagician)

greatMagicians = []
def makeGreat(magicians):
    while magicians:
        magician = magicians.pop()
        greatMagician = magician + " the Great"
        greatMagicians.append(greatMagician)
    for greatMagician in greatMagicians:
        magicians.append(greatMagician)


magicians = ["David Copperfield", "David Blaine", "Criss Angel"]
makeGreat(magicians[:])
showMagicians(magicians, greatMagicians)
```

```
David Copperfield
David Blaine
Criss Angel
Criss Angel the Great
David Blaine the Great
David Copperfield the Great
```

Exercise 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 [ ]:
```
def makeSandwich(*ingredients):
    print("\n Making a sandwich with the following ingredients:")
    for ingredient in ingredients:
        print("- " + ingredient)
    print("Your sandwich is ready!")

makeSandwich("chicken", "lettuce", "tomato")
makeSandwich("pepperoni", "cheese")
makeSandwich("salami", "cheese", "onion", "tomato")
```

```
 Making a sandwich with the following ingredients:
- chicken
- lettuce
- tomato
Your sandwich is ready!

 Making a sandwich with the following ingredients:
- pepperoni
- cheese
Your sandwich is ready!

 Making a sandwich with the following ingredients:
- salami
- cheese
- onion
- tomato
Your sandwich is ready!
```

Exercise 8.13: User Profile.

Start with a copy of user_profile.py from page 153. Build a profile of yourself by calling `build_profile()`, using your first and last names and three other key-value pairs that describe you.

In [ ]:
```
def build_profile(first, last, **user_info):
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile

user_profile = build_profile('Muid', 'Bin Aslam', university='Bahria University', m
print(user_profile)
```

```
{'first_name': 'Muid', 'last_name': 'Bin Aslam', 'university': 'Bahria Universit
y', 'major': 'Computer Science', 'fieldOfInterest': 'Data Science'}
```

Exercise 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 [ ]:
```
def makeCar(manufacturer, model, **options):
    car = {}
    car['manufacturer'] = manufacturer
    car['model'] = model
    for key, value in options.items():
        car[key] = value
    return car

car = makeCar('subaru', 'outback', color='blue', tow_package=True)
print(car)

car=makeCar('ferrari', 'f8', color='red', year=2020, price=300000)
print(car)
```

```
{'manufacturer': 'subaru', 'model': 'outback', 'color': 'blue', 'tow_package': Tru
e}
{'manufacturer': 'ferrari', 'model': 'f8', 'color': 'red', 'year': 2020, 'price':
300000}
```