Functions – Exercises and Answers

Exercise 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.

**def** **display\_message**():

"""Display a message about what I'm learning."""

msg **=** "I'm learning to store code in functions."

**print**(msg)

display\_message()

Output:

I'm learning to store code in functions.

Exercise 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.

**def** **favorite\_book**(title):

"""Display a message about someone's favorite book."""

**print**(title **+** " is one of my favorite books.")

favorite\_book('The Abstract Wild')

Output:

The Abstract Wild is one of my favorite books.

Exercise 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.

**def** **make\_shirt**(size, message):

"""Summarize the shirt that's going to be made."""

**print**("\nI'm going to make a " **+** size **+** " t-shirt.")

**print**('It will say, "' **+** message **+** '"')

make\_shirt('large', 'I love Python!')

make\_shirt(message**=**"Readability counts.", size**=**'medium')

Output:

I'm going to make a large t-shirt.

It will say, "I love Python!"

I'm going to make a medium t-shirt.

It will say, "Readability counts."

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 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.

**def** **make\_shirt**(size**=**'large', message**=**'I love Python!'):

"""Summarize the shirt that's going to be made."""

**print**("\nI'm going to make a " **+** size **+** " t-shirt.")

**print**('It will say, "' **+** message **+** '"')

make\_shirt()

make\_shirt(size**=**'medium')

make\_shirt('small', 'Programmers are loopy.')

Output:

I'm going to make a large t-shirt.

It will say, "I love Python!"

I'm going to make a medium t-shirt.

It will say, "I love Python!"

I'm going to make a small t-shirt.

It will say, "Programmers are loopy."

Exercise 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.

**def** **describe\_city**(city, country**=**'chile'):

"""Describe a city."""

msg **=** city**.**title() **+** " is in " **+** country**.**title() **+** "."

**print**(msg)

describe\_city('santiago')

describe\_city('reykjavik', 'iceland')

describe\_city('punta arenas')

Output:

Santiago is in Chile.

Reykjavik is in Iceland.

Punta Arenas is in Chile.

Exercise 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.

**def** **city\_country**(city, country):

"""Return a string like 'Santiago, Chile'."""

**return**(city**.**title() **+** ", " **+** country**.**title())

city **=** city\_country('santiago', 'chile')

**print**(city)

city **=** city\_country('ushuaia', 'argentina')

**print**(city)

city **=** city\_country('longyearbyen', 'svalbard')

**print**(city)

Output:

Santiago, Chile

Ushuaia, Argentina

Longyearbyen, Svalbard

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 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 nubmer 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 nubmer of tracks on an album.

Simple version:

**def** **make\_album**(artist, title):

"""Build a dictionary containing information about an album."""

album\_dict **=** {

'artist': artist**.**title(),

'title': title**.**title(),

}

**return** album\_dict

album **=** make\_album('metallica', 'ride the lightning')

**print**(album)

album **=** make\_album('beethoven', 'ninth symphony')

**print**(album)

album **=** make\_album('willie nelson', 'red-headed stranger')

**print**(album)

Output:

{'title': 'Ride The Lightning', 'artist': 'Metallica'}

{'title': 'Ninth Symphony', 'artist': 'Beethoven'}

{'title': 'Red-Headed Stranger', 'artist': 'Willie Nelson'}

With tracks:

**def** **make\_album**(artist, title, tracks**=**0):

"""Build a dictionary containing information about an album."""

album\_dict **=** {

'artist': artist**.**title(),

'title': title**.**title(),

}

**if** tracks:

album\_dict['tracks'] **=** tracks

**return** album\_dict

album **=** make\_album('metallica', 'ride the lightning')

**print**(album)

album **=** make\_album('beethoven', 'ninth symphony')

**print**(album)

album **=** make\_album('willie nelson', 'red-headed stranger')

**print**(album)

album **=** make\_album('iron maiden', 'piece of mind', tracks**=**8)

**print**(album)

Output:

{'artist': 'Metallica', 'title': 'Ride The Lightning'}

{'artist': 'Beethoven', 'title': 'Ninth Symphony'}

{'artist': 'Willie Nelson', 'title': 'Red-Headed Stranger'}

{'tracks': 8, 'artist': 'Iron Maiden', 'title': 'Piece Of Mind'}

Exercise 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.

**def** **make\_album**(artist, title, tracks**=**0):

"""Build a dictionary containing information about an album."""

album\_dict **=** {

'artist': artist**.**title(),

'title': title**.**title(),

}

**if** tracks:

album\_dict['tracks'] **=** tracks

**return** album\_dict

*# Prepare the prompts.*

title\_prompt **=** "\nWhat album are you thinking of? "

artist\_prompt **=** "Who's the artist? "

*# Let the user know how to quit.*

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

**while** True:

title **=** input(title\_prompt)

**if** title **==** 'quit':

**break**

artist **=** input(artist\_prompt)

**if** artist **==** 'quit':

**break**

album **=** make\_album(artist, title)

**print**(album)

**print**("\nThanks for responding!")

Output:

Enter 'quit' at any time to stop.

What album are you thinking of? **number of the beast**

Who's the artist? **iron maiden**

{'artist': 'Iron Maiden', 'title': 'Number Of The Beast'}

What album are you thinking of? **touch of class**

Who's the artist? **angel romero**

{'artist': 'Angel Romero', 'title': 'Touch Of Class'}

What album are you thinking of? **rust in peace**

Who's the artist? **megadeth**

{'artist': 'Megadeth', 'title': 'Rust In Peace'}

What album are you thinking of? **quit**

Thanks for responding!

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 9: Magicians

Make a list of magician’s names. Pass the list to a function called show\_magicians(), wich prints the name of each magician in the list.

**def** **show\_magicians**(magicians):

"""Print the name of each magician in the list."""

**for** magician **in** magicians:

**print**(magician**.**title())

magicians **=** ['harry houdini', 'david blaine', 'teller']

show\_magicians(magicians)

Output:

Harry Houdini

David Blaine

Teller

Exercise 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.

**def** **show\_magicians**(magicians):

"""Print the name of each magician in the list."""

**for** magician **in** magicians:

**print**(magician)

**def** **make\_great**(magicians):

"""Add 'the Great!' to each magician's name."""

*# Build a new list to hold the great musicians.*

great\_magicians **=** []

*# Make each magician great, and add it to great\_magicians.*

**while** magicians:

magician **=** magicians**.**pop()

great\_magician **=** magician **+** ' the Great'

great\_magicians**.**append(great\_magician)

*# Add the great magicians back into magicians.*

**for** great\_magician **in** great\_magicians:

magicians**.**append(great\_magician)

magicians **=** ['Harry Houdini', 'David Blaine', 'Teller']

show\_magicians(magicians)

**print**("\n")

make\_great(magicians)

show\_magicians(magicians)

Output:

Harry Houdini

David Blaine

Teller

Teller the Great

David Blaine the Great

Harry Houdini the Great

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 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.

**def** **show\_magicians**(magicians):

"""Print the name of each magician in the list."""

**for** magician **in** magicians:

**print**(magician)

**def** **make\_great**(magicians):

"""Add 'the Great!' to each magician's name."""

*# Build a new list to hold the great musicians.*

great\_magicians **=** []

*# Make each magician great, and add it to great\_magicians.*

**while** magicians:

magician **=** magicians**.**pop()

great\_magician **=** magician **+** ' the Great'

great\_magicians**.**append(great\_magician)

*# Add the great magicians back into magicians.*

**for** great\_magician **in** great\_magicians:

magicians**.**append(great\_magician)

**return** magicians

magicians **=** ['Harry Houdini', 'David Blaine', 'Teller']

show\_magicians(magicians)

**print**("\nGreat magicians:")

great\_magicians **=** make\_great(magicians[:])

show\_magicians(great\_magicians)

**print**("\nOriginal magicians:")

show\_magicians(magicians)

Output:

Harry Houdini

David Blaine

Teller

Great magicians:

Teller the Great

David Blaine the Great

Harry Houdini the Great

Original magicians:

Harry Houdini

David Blaine

Teller

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 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 sandiwch that is being ordered. Call the function three tiems, using a different number of arguments each time.

**def** **make\_sandwich**(**\***items):

"""Make a sandwich with the given items."""

**print**("\nI'll make you a great sandwich:")

**for** item **in** items:

**print**(" ...adding " **+** item **+** " to your sandwich.")

**print**("Your sandwich is ready!")

make\_sandwich('roast beef', 'cheddar cheese', 'lettuce', 'honey dijon')

make\_sandwich('turkey', 'apple slices', 'honey mustard')

make\_sandwich('peanut butter', 'strawberry jam')

Output:

I'll make you a great sandwich:

...adding roast beef to your sandwich.

...adding cheddar cheese to your sandwich.

...adding lettuce to your sandwich.

...adding honey dijon to your sandwich.

Your sandwich is ready!

I'll make you a great sandwich:

...adding turkey to your sandwich.

...adding apple slices to your sandwich.

...adding honey mustard to your sandwich.

Your sandwich is ready!

I'll make you a great sandwich:

...adding peanut butter to your sandwich.

...adding strawberry jam to your sandwich.

Your sandwich is ready!

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise 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.

**def** **make\_car**(manufacturer, model, **\*\***options):

"""Make a dictionary representing a car."""

car\_dict **=** {

'manufacturer': manufacturer**.**title(),

'model': model**.**title(),

}

**for** option, value **in** options**.**items():

car\_dict[option] **=** value

**return** car\_dict

my\_outback **=** make\_car('subaru', 'outback', color**=**'blue', tow\_package**=**True)

**print**(my\_outback)

my\_accord **=** make\_car('honda', 'accord', year**=**1991, color**=**'white',

headlights**=**'popup')

**print**(my\_accord)

Output:

{'manufacturer': 'Subaru', 'color': 'blue', 'tow\_package': True, 'model': 'Outback'}

{'year': 1991, 'manufacturer': 'Honda', 'color': 'white', 'headlights': 'popup', 'model': 'Accord'}

[top](https://ehmatthes.github.io/pcc/solutions/chapter_8.html)

Exercise15: Printing Models

Put the functions for the example *printing\_models.py* in a separate file called *printing\_functions.py*. Write an import statement at the top of *printing\_models.py*, and modify the file to use the imported functions.

***Note:****The text refers to print\_models.py, but it should say printing\_models.py.*

*printing\_functions.py:*

"""Functions related to printing 3d models."""

**def** **print\_models**(unprinted\_designs, completed\_models):

"""

Simulate printing each design, until there are none left.

Move each design to completed\_models after printing.

"""

**while** unprinted\_designs:

current\_design **=** unprinted\_designs**.**pop()

*# Simulate creating a 3d print from the design.*

**print**("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)

*printing\_models.py:*

import printing\_functions **as** pf

unprinted\_designs **=** ['iphone case', 'robot pendant', 'dodecahedron']

completed\_models **=** []

pf**.**print\_models(unprinted\_designs, completed\_models)

pf**.**show\_completed\_models(completed\_models)

Output:

Printing model: dodecahedron

Printing model: robot pendant

Printing model: iphone case

The following models have been printed:

dodecahedron

robot pendant

iphone case