Name : Syed Zulfiqar Hassan
I'd : 14937
Subject : Modern programming language
Submitted to : Sir Fahim Ullah

Task 3-1: Store the names of a few of your friends in a list called names. Print
each person's name by accessing each element in the list, one at a time.

Solution:

```
Friend_Names=['hameed','salman','karim','sauood']

print(Friend_Names)
```

Output:

```
['hameed', 'salman', 'karim', 'sauood']
```

Task 3-2: Start with the list you used in Exercise 3-1, but instead of just
printing each person's name, print a message to them. The text of each message
should be the same, but each message should be personalized with the
person's name.

Solution:

```
Friend_Names=['hameed','salman','karim','sauood']

message="My room mate is " + Friend_Names[0].title() + "." + Friend_Names[1].title() + " and " + Friend_Names[2].title() + " are my Classmates." + Friend_Names[3].title() + " is my best Friend."

print(message)
```

Output:

```
My room mate is Hameed.Salman and Karim are my Classmates.Sauood is my best Friend.
```

Task 3-3: Think of your favorite mode of transportation, such as a motorcycle or a car, and make a list that stores several examples. Use your list
to print a series of statements about these items, such as “I would like to own a
Honda motorcycle.

Solution:

```
Subjects=['DLD','DS','OB','MPL']

MS=Subjects[3] + " is my best  subject and this subject teaches us Sir Faheem." + Subjects[0] + "," + Subjects[1] + " and " + Subjects[2] + " abbreviate, ' Digital and Logical Data', 'Data Structure' and 'Organizational Behaviour' respectively."

print(MS)
```

Output:

```
MPL is my best  subject and this subject teaches us Sir
```

Faheem.DLD,DS and OB abbreviate, ' Digital and Logical Data', 'Data Structure' and 'Organizational Behaviour' respectively.

Task 3-4: If you could invite anyone, living or deceased, to dinner, who would you invite? Make a list that includes at least three people you'd like to invite to dinner. Then use your list to print a message to each person, inviting them to dinner.

Solution:

```
list=['Doctor','Teacher','Engineer']

MSG=" I invite you " + list[0] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[1] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[2] + " to Dinner at 2:00PM in Peshawar Thanks."

print(MSG)
```

Output:

```
I invite you Doctor to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Teacher to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Engineer to Dinner at 2:00PM in Peshawar Thanks.
```

Task 3-5: You just heard that one of your guests can't make the dinner, so you need to send out a new set of invitations. You'll have to think of someone else to invite.

• Start with your program from Exercise 3-4. Add a print statement at the

end of your program stating the name of the guest who can’t make it.
• Modify your list, replacing the name of the guest who can’t make it with
the name of the new person you are inviting.
• Print a second set of invitation messages, one for each person who is still
in your list.

Solution:

```
list=['Doctor','Teacher','Engineer']

list[1]='Police'

print(list)

MSG=" I invite you " + list[0] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[1] + " to Dinner at 2:00PM in PeshawarThanks." + "\n I invite you " + list[2] + " to Dinner at 2:00PM in Peshawar Thanks."

print(MSG)
```

Output:

```
['Doctor', 'Police', 'Engineer']

 I invite you Doctor to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Police to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Engineer to Dinner at 2:00PM in Peshawar Thanks.
```

Task 3-6: You just found a bigger dinner table, so now more space is

available. Think of three more guests to invite to dinner.
• Start with your program from Exercise 3-4 or Exercise 3-5. Add a print
statement to the end of your program informing people that you found a
bigger dinner table.
• Use insert() to add one new guest to the beginning of your list.
• Use insert() to add one new guest to the middle of your list.
• Use append() to add one new guest to the end of your list.
• Print a new set of invitation messages, one for each person in your list.

Solution:

```
list=['Doctor','Teacher','Engineer']

list.insert(0,'Businessman')

print(list)

list.insert(2,'Watchman')

print(list)

list.append('Gardener')

print(list)

MSG=" I invite you " + list[0] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[1] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[2] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[3] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[4] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[5] + " to Dinner at 2:00PM in Peshawar Thanks."
```

```
print(MSG)
```

Output:

```
['Businessman', 'Doctor', 'Teacher', 'Engineer']

['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer']

['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer',
'Gardener']

 I invite you Businessman to Dinner at 2:00PM in Peshawar
Thanks.

 I invite you Doctor to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Watchman to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Teacher to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Engineer to Dinner at 2:00PM in Peshawar Thanks.

 I invite you Gardener to Dinner at 2:00PM in Peshawar Thanks.
```

Task 3-7: You just found out that your new dinner table won't arrive in time for the dinner, and you have space for only two guests.
• Start with your program from Exercise 3-6. Add a new line that prints a
message saying that you can invite only two people for dinner.
• Use pop() to remove guests from your list one at a time until only two
names remain in your list. Each time you pop a name from your list, print
a message to that person letting them know you're sorry you can't invite

them to dinner.
• Print a message to each of the two people still on your list, letting them
know they’re still invited.
• Use del to remove the last two names from your list, so you have an empty
list. Print your list to make sure you actually have an empty list at the end
of your program.

Solution:

```
list=['Doctor','Teacher','Engineer']

list.insert(0,'Businessman')

print(list)

list.insert(2,'Watchman')

print(list)

list.append('Gardener')

print(list)

line="I can invite only two people for Dinner."

print(line)

print(list[5] +   " I am sorry, I can't invite you to Dinner.")

popped_list =list.pop()

print(list)

print(popped_list)
```

```
print(list[4] +   " I am sorry, I can't invite you to Dinner.")

popped_list =list.pop()

print(list)

print(popped_list)

print(list[3] +   " I am sorry, I can't invite you to Dinner.")

popped_list=list.pop()

print(list)

print(popped_list)

print(list[2] +   " I am sorry, I can't invite you to Dinner.")

popped_list=list.pop()

print(list)

print(popped_list)

MSG=" I invite you " + list[0] + " to Dinner at 2:00PM in Peshawar Thanks." + "\n I invite you " + list[1] + " to Dinner at 2:00PM in Peshawar Thanks."

print(MSG)
```

Output:

```
['Businessman', 'Doctor', 'Teacher', 'Engineer']

['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer']

['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer', 'Gardener']
```

I can invite only two people for Dinner.

Gardener I am sorry, I can't invite you to Dinner.

['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer']

Gardener

Engineer I am sorry, I can't invite you to Dinner.

['Businessman', 'Doctor', 'Watchman', 'Teacher']

Engineer

Teacher I am sorry, I can't invite you to Dinner.

['Businessman', 'Doctor', 'Watchman']

Teacher

Watchman I am sorry, I can't invite you to Dinner.

['Businessman', 'Doctor']

Watchman

I invite you Businessman to Dinner at 2:00PM in Peshawar Thanks.

I invite you Doctor to Dinner at 2:00PM in Peshawar Thanks.

Task 3-8: Think of at least five places in the world you'd like to visit.
• Store the locations in a list. Make sure the list is not in alphabetical order.
• Print your list in its original order. Don't worry about printing the list neatly,
just print it as a raw Python list.

• Use sorted() to print your list in alphabetical order without modifying the
actual list.
• Show that your list is still in its original order by printing it.
• Use sorted() to print your list in reverse alphabetical order without changing
the order of the original list.
• Show that your list is still in its original order by printing it again.
• Use reverse() to change the order of your list. Print the list to show that its
order has changed.
• Use reverse() to change the order of your list again. Print the list to show
it’s back to its original order.
• Use sort() to change your list so it’s stored in alphabetical order. Print the
list to show that its order has been changed.
• Use sort() to change your list so it’s stored in reverse alphabetical order.
Print the list to show that its order has changed.

Solution:

```
Places=['Japan','India','Pakistan','America']

print(Places)

print("\nHere is the sorted list.")

print(sorted(Places))

print("\nHere is the original list.")

print(Places)
```

```
print("\nHere is the sorted list.")
print(sorted(Places,reverse=True)
print("\nHere is the original list again.")
print(Places)
print("\nHere the list in reverse order.")
Places.reverse()
print(Places)
print("\nHere the list in reverse order again.")
Places.reverse()
print(Places)
print("\nHere is the sort list.")
Places.sort()
print(Places)
print("\nHere is the sort list in reverse.")
Places.sort(reverse=True)
print(Places)
```

Output:

```
['Japan', 'India', 'Pakistan', 'America']

Here is the sorted list.
```

['America', 'India', 'Japan', 'Pakistan']

Here is the original list.

['Japan', 'India', 'Pakistan', 'America']

Here is the sorted list.

['Pakistan', 'Japan', 'India', 'America']

Here is the original list again.

['Japan', 'India', 'Pakistan', 'America']

Here the list in reverse order.

['America', 'Pakistan', 'India', 'Japan']

Here the list in reverse order again.

['Japan', 'India', 'Pakistan', 'America']

Here is the sort list.

['America', 'India', 'Japan', 'Pakistan']

Here is the sort list in reverse.

['Pakistan', 'Japan', 'India', 'America']

Task 3-9: Working with one of the programs from Exercises 3-4 through 3-7 (page 46), use len() to print a message indicating the number
of people you are inviting to dinner.

Solution:

```
list=['Businessman', 'Doctor', 'Watchman', 'Teacher', 'Engineer', 'Gardener']

print(len(list))

msge="The number of people I invited to Dinner are " + str(len(list)) + "."

print(msge)
```

Output:

6

The number of people I invited to Dinner are 6.

Task 3-10: If you haven't received an index error in one of your programs yet, try to make one happen. Change an index in one of your Programs to produce an index error. Make sure you correct the error before closing the program.
Solution:

```
list=['Businessman', 'Doctor', 'Watchman']

print(list[3])
```

Correction of error:

```
list=['Businessman', 'Doctor', 'Watchman']

print(list[2])
```

Output:

```
Traceback (most recent call last):

  File "C:/Users/X61s/AppData/Local/Programs/Python/8.py", line 2, in <module>

    print(list[3])

IndexError: list index out of range
```

Correction of error:

```
Watchman
```

Task 4-1:Think of at least three kinds of your favorite pizza. Store these pizza names in a list, and then use a for loop to print the name of each pizza.
• Modify your for loop to print a sentence using the name of the pizza instead of printing just the name of the pizza. For each pizza you should have one line of output containing a simple statement like *I like pepperoni pizza*.
• Add a line at the end of your program, outside the for loop, that states how much you like pizza. The output should consist of three or more lines about the kinds of pizza you like and then an additional sentence, such as *I really love pizza!*

Solution:

```
Pizzas=[‘california’,’broadway’,’penny’]

for pizza in pizzas:

   print(pizza)

   msg=pizza.title() + " is very sweet."

   print(msg)
```

```
print("I like to go to eat pizza with my friends.")
```

Output:

```
california

California is very sweet.

broadway

Broadway is very sweet.

penny

Penny is very sweet.

I like to go to eat pizza with my friends.
```

Task 4-2: Think of at least three different animals that have a common characteristic. Store the names of these animals in a list, and then use a for loop to print out the name of each animal.
• Modify your program to print a statement about each animal, such as *A dog would make a great pet.*
• Add a line at the end of your program stating what these animals have in common. You could print a sentence such as *Any of these animals would make a great pet!*

Solution:

```
animals=['cow','goat','horse']

for animal in animals:

    print(animal)

statement="It is known as the " + animals[0].title() + " is our Mother  in India." + "\nThe horn of the " + animals[1].title() + " is used to make spoons." + "\n" + animals[2].title() + " is a very powerful animal."

print(statement)

print("All these animals have four legs, they eat grass and they give milk.")
```

Output:

```
cow
```

goat

horse

It is known as the Cow is our Mother in India.

The horn of the Goat is used to make spoons.

Horse is a very powerful animal.

All these animals have four legs, they eat grass and they give milk.

Task 4-3: Use a for loop to print the numbers from 1 to 20, inclusive.

Solution:

```
for value in range(1,21):
    print(value)
```

Output:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

Task 4-4: Make a list of the numbers from one to one million, and then use a for loop to print the numbers. (If the output is taking too long, stop it by pressing ctrl-C or by closing the output window.)

Solution:

```
numbers=list(range(1,1000000))

print(numbers)
```

Output:

Its output is too long.

Task 4-5: Make a list of the numbers from one to one million, and then use min() and max() to make sure your list actually starts at one and ends at one million. Also, use the sum() function to see how quickly Python can add a million numbers.

Solution:

```
>>> numbers=list(range(1,1000001))

>>> min(numbers)

1

>>> max(numbers)

1000000

>>> sum(numbers)
```

500000500000

Task 4-6: Use the third argument of the range() function to make a list of the odd numbers from 1 to 20. Use a for loop to print each number.

Solution:

```
odd_numbers=list(range(1,20,2))

for value in range(1,20,2):

   odd_numbers.append(value+2)

print(odd_numbers)
```

Output:

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]

Task 4-7: Make a list of the multiples of 3 from 3 to 30. Use a for loop to print the numbers in your list.

Solution:

```
list=[]

for value in range(1,11):

   list.append(value*3)

print(list)
```

Output:

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30]

Task 4-8: A number raised to the third power is called a *cube*. For example, the cube of 2 is written as 2**3 in Python. Make a list of the first 10 cubes (that is, the cube of each integer from 1 through 10), and use a for loop to print out the value of each cube.

Solution:

```
cubes=[]

for value in range(1,11):
```

```
    cubes.append(value**3)

print(cubes)
```

Output:

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Task 4-9: Use a list comprehension to generate a list of the first 10 cubes.

Solution:

```
cubes=[value**3 for value in range(1,11)]

print(cubes)
```

Output:

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

4-11: My Pizzas, Your Pizzas

Start with your program from Exercise 4-1 (page 60). Make a copy of the list of pizzas, and call it `friend_pizzas`. Then, do the following:

- Add a new pizza to the original list.
- Add a different pizza to the list `friend_pizzas`.
- Prove that you have two separate lists. Print the message, *My favorite pizzas are:*, and then use a `for` loop to print the first list. Print the message, *My friend's favorite pizzas are:*, and then use a `for` loop to print the second list. Make sure each new pizza is stored in the appropriate list.

```
favorite_pizzas = ['pepperoni', 'hawaiian', 'veggie']
friend_pizzas = favorite_pizzas[:]

favorite_pizzas.append("meat lover's")
friend_pizzas.append('pesto')

print("My favorite pizzas are:")
for pizza in favorite_pizzas:
    print("- " + pizza)

print("\nMy friend's favorite pizzas are:")
for pizza in friend_pizzas:
    print("- " + pizza)
```

Output:

```
My favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- meat lover's

My friend's favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- pesto
```

top

4-13: Buffet

A buffet-style restaurant offers only five basic foods. Think of five simple foods, and store them in a tuple.

- Use a `for` loop to print each food the restaurant offers.
- Try to modify one of the items, and make sure that Python rejects the change.
- The restaurant changes its menu, replacing two of the items with different foods. Add a block of code that rewrites the tuple, and then use a `for` loop to print each of the items on the revised menu.

```
menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'salmon burger', 'crab cakes',
    )

print("You can choose from the following menu items:")
for item in menu_items:
    print("- " + item)

menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'black cod tips', 'king crab legs',
    )

print("\nOur menu has been updated.")
print("You can now choose from the following items:")
for item in menu_items:
    print("- " + item)
```

Output:

```
You can choose from the following menu items:
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- salmon burger
- crab cakes

Our menu has been updated.
You can now choose from the following items:
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- black cod tips
- king crab legs
```

6-1: Person

Use a dictionary to store information about a person you know. Store their first name, last name, age, and the city in which they live. You should have keys such as `first_name`, `last_name`, `age`, and `city`. Print each piece of information stored in your dictionary.

```
person = {
    'first_name': 'eric',
    'last_name': 'matthes',
    'age': 43,
    'city': 'sitka',
    }

print(person['first_name'])
print(person['last_name'])
print(person['age'])
print(person['city'])
```

Output:

```
eric
matthes
43
sitka
```

top

6-2: Favorite Numbers

Use a dictionary to store people's favorite numbers. Think of five names, and use them as keys in your dictionary. Think of a favorite number for each person, and store each as a value in your dictionary. Print each person's name and their favorite number. For even more fun, poll a few friends and get some actual data for your program.

```
favorite_numbers = {
    'mandy': 42,
    'micah': 23,
    'gus': 7,
    'hank': 1000000,
    'maggie': 0,
    }

num = favorite_numbers['mandy']
print("Mandy's favorite number is " + str(num) + ".")

num = favorite_numbers['micah']
print("Micah's favorite number is " + str(num) + ".")

num = favorite_numbers['gus']
print("Gus's favorite number is " + str(num) + ".")

num = favorite_numbers['hank']
print("Hank's favorite number is " + str(num) + ".")

num = favorite_numbers['maggie']
print("Maggie's favorite number is " + str(num) + ".")
```

Output:

```
Mandy's favorite number is 42.
Micah's favorite number is 23.
Gus's favorite number is 7.
Hank's favorite number is 1000000.
Maggie's favorite number is 0.
```

top

6-3: Glossary

A Python dictionary can be used to model an actual dictionary. However, to avoid confusion, let's call it a glossary.

- Think of five programming words you've learned about in the previous chapters. Use these words as the keys in your glossary, and store their meanings as values.
- Print each word and its meaning as neatly formatted output. You might print the word followed by a colon and then its meaning, or print the word

on one line and then print its meaning indented on a second line. Use the newline character (`'\n'`) to insert a blank line between each word-meaning pair in your output.

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    }

word = 'string'
print("\n" + word.title() + ": " + glossary[word])

word = 'comment'
print("\n" + word.title() + ": " + glossary[word])

word = 'list'
print("\n" + word.title() + ": " + glossary[word])

word = 'loop'
print("\n" + word.title() + ": " + glossary[word])

word = 'dictionary'
print("\n" + word.title() + ": " + glossary[word])
```

Output:

```
String: A series of characters.

Comment: A note in a program that the Python interpreter ignores.

List: A collection of items in a particular order.

Loop: Work through a collection of items, one at a time.

Dictionary: A collection of key-value pairs.
```

top

6-4: Glossary 2

Now that you know how to loop through a dictionary, clean up the code from Exercise 6-3 (page 102) by replacing your series of `print` statements with a loop that runs through the dictionary's keys and values. When you're sure that your loop works, add five more Python terms to your glossary. When you run your program again, these new words and meanings should automatically be included in the output.

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    'key': 'The first item in a key-value pair in a dictionary.',
    'value': 'An item associated with a key in a dictionary.',
    'conditional test': 'A comparison between two values.',
    'float': 'A numerical value with a decimal component.',
    'boolean expression': 'An expression that evaluates to True or False.',
    }

for word, definition in glossary.items():
    print("\n" + word.title() + ": " + definition)
```

Output:

```
Dictionary: A collection of key-value pairs.

String: A series of characters.

Boolean Expression: An expression that evaluates to True or False.

Comment: A note in a program that the Python interpreter ignores.

Value: An item associated with a key in a dictionary.

Loop: Work through a collection of items, one at a time.

List: A collection of items in a particular order.

Conditional Test: A comparison between two values.

Key: The first item in a key-value pair in a dictionary.

Float: A numerical value with a decimal component.
```

top

6-5: Rivers

Make a dictionary containing three major rivers and the country each river runs through. One key-value pair might be `'nile': 'egypt'`.

- Use a loop to print a sentence about each river, such as *The Nile runs through Egypt.*
- Use a loop to print the name of each river included in the dictionary.
- Use a loop to print the name of each country included in the dictionary.

```
rivers = {
    'nile': 'egypt',
    'mississippi': 'united states',
    'fraser': 'canada',
    'kuskokwim': 'alaska',
    'yangtze': 'china',
    }

for river, country in rivers.items():
    print("The " + river.title() + " flows through " + country.title() + ".")

print("\nThe following rivers are included in this data set:")
for river in rivers.keys():
    print("- " + river.title())

print("\nThe following countries are included in this data set:")
for country in rivers.values():
    print("- " + country.title())
```

Output*:

```
The Mississippi flows through United States.
The Yangtze flows through China.
The Fraser flows through Canada.
The Nile flows through Egypt.
The Kuskokwim flows through Alaska.

The following rivers are included in this data set:
- Mississippi
- Yangtze
- Fraser
- Nile
- Kuskokwim

The following countries are included in this data set:
- United States
- China
- Canada
- Egypt
- Alaska
```

*Sometimes we like to think of Alaska as our own separate country.

top

6-6: Polling

Use the code in *favorite_languages.py* (page 104).

- Make a list of people who should take the favorite languages poll. Include some names that are already in the dictionary and some that are not.

- Loop through the list of people who should take the poll. If they have already taken the poll, print a message thanking them for responding. If they have not yet taken the poll, print a message inviting them to take the poll.

```
favorite_languages = {
   'jen': 'python',
   'sarah': 'c',
   'edward': 'ruby',
   'phil': 'python',
   }

for name, language in favorite_languages.items():
   print(name.title() + "'s favorite language is " +
      language.title() + ".")

print("\n")

coders = ['phil', 'josh', 'david', 'becca', 'sarah', 'matt', 'danielle']
for coder in coders:
   if coder in favorite_languages.keys():
      print("Thank you for taking the poll, " + coder.title() + "!")
   else:
      print(coder.title() + ", what's your favorite programming language?")
```

Output:

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Phil's favorite language is Python.
Edward's favorite language is Ruby.


Thank you for taking the poll, Phil!
Josh, what's your favorite programming language?
David, what's your favorite programming language?
Becca, what's your favorite programming language?
Thank you for taking the poll, Sarah!
Matt, what's your favorite programming language?
Danielle, what's your favorite programming language?
```

top

6-7: People

Start with the program you wrote for Exercise 6-1 (page 102). Make two new dictionaries representing different people, and store all three dictionaries in a list called `people`. Loop through your list of people. As you loop through the list, print everything you know about each person.

```
# Make an empty list to store people in.
people = []

# Define some people, and add them to the list.
person = {
   'first_name': 'eric',
   'last_name': 'matthes',
   'age': 43,
   'city': 'sitka',
   }
people.append(person)

person = {
   'first_name': 'ever',
   'last_name': 'matthes',
   'age': 5,
   'city': 'sitka',
   }
people.append(person)

person = {
   'first_name': 'willie',
   'last_name': 'matthes',
   'age': 8,
   'city': 'sitka',
   }
people.append(person)

# Display all of the information in the dictionary.
for person in people:
   name = person['first_name'].title() + " " + person['last_name'].title()
   age = str(person['age'])
   city = person['city'].title()

   print(name + ", of " + city + ", is " + age + " years old.")
```

Output:

```
Eric Matthes, of Sitka, is 43 years old.
Ever Matthes, of Sitka, is 5 years old.
Willie Matthes, of Sitka, is 8 years old.
```

top

6-8: Pets

Make several dictionaries, where the name of each dictionary is the name of a pet. In each dictionary, include the kind of animal and the owner's name. Store these dictionaries in a list called `pets`. Next, loop through your list and as you do print everything you know about each pet.

Note: When I decided to post solutions and wrote complete programs to solve each exercise, I realized this problem was not as well phrased as it should have been. It doesn't really make sense to name each dictionary for the pet it describes; that information should really be included in the dictionary, rather than being used as the name of the dictionary. This solution reflects that approach.

```
# Make an empty list to store the pets in.
pets = []

# Make individual pets, and store each one in the list.
pet = {
    'animal type': 'python',
    'name': 'john',
    'owner': 'guido',
    'weight': 43,
    'eats': 'bugs',
}
pets.append(pet)

pet = {
    'animal type': 'chicken',
    'name': 'clarence',
    'owner': 'tiffany',
    'weight': 2,
    'eats': 'seeds',
}
pets.append(pet)

pet = {
    'animal type': 'dog',
    'name': 'peso',
    'owner': 'eric',
    'weight': 37,
    'eats': 'shoes',
}
pets.append(pet)

# Display information about each pet.
for pet in pets:
    print("\nHere's what I know about " + pet['name'].title() + ":")
    for key, value in pet.items():
        print("\t" + key + ": " + str(value))
```

Output:

```
Here's what I know about John:
    weight: 43
    animal type: python
    name: john
    owner: guido
    eats: bugs
```

```
Here's what I know about Clarence:
   weight: 2
   animal type: chicken
   name: clarence
   owner: tiffany
   eats: seeds

Here's what I know about Peso:
   weight: 37
   animal type: dog
   name: peso
   owner: eric
   eats: shoes
```

top

6-9: Favorite Places

Make a dictionary called `favorite_places`. Think of three names to use as keys in the dictionary, and store one to three favorite places for each person. To make this exericse a bit more interesting, ask some friends to name a few of their favorite places. Loop through the dictionary, and print each person's name and their favorite places.

```
favorite_places = {
   'eric': ['bear mountain', 'death valley', 'tierra del fuego'],
   'erin': ['hawaii', 'iceland'],
   'ever': ['mt. verstovia', 'the playground', 'south carolina']
   }

for name, places in favorite_places.items():
   print("\n" + name.title() + " likes the following places:")
   for place in places:
      print("- " + place.title())
```

Output:

```
Ever likes the following places:
- Mt. Verstovia
- The Playground
- South Carolina

Erin likes the following places:
- Hawaii
- Iceland

Eric likes the following places:
- Bear Mountain
- Death Valley
- Tierra Del Fuego
```

top

6-10: Favorite Numbers

Modify your program from Exercise 6-2 (page 102) so each person can have more than one favorite number. Then print each person's name along with their favorite numbers.

```
favorite_numbers = {
    'mandy': [42, 17],
    'micah': [42, 39, 56],
    'gus': [7, 12],
    }

for name, numbers in favorite_numbers.items():
    print("\n" + name.title() + " likes the following numbers:")
    for number in numbers:
        print("  " + str(number))
```

Output:

```
Micah likes the following numbers:
  42
  39
  56

Mandy likes the following numbers:
  42
  17

Gus likes the following numbers:
  7
  12
```

top

6-11: Cities

Make a dictionary called `cities`. Use the names of three cities as keys in your dictionary. Create a dictionary of information about each city and include the country that the city is in, its approximate population, and one fact about that city. The keys for each city's dictionary should be something like `country`, `population`, and `fact`. Print the name of each city and all of the information you have stored about it.

```
cities = {
    'santiago': {
        'country': 'chile',
```

```
        'population': 6158080,
        'nearby mountains': 'andes',
        },
    'talkeetna': {
        'country': 'alaska',
        'population': 876,
        'nearby mountains': 'alaska range',
        },
    'kathmandu': {
        'country': 'nepal',
        'population': 1003285,
        'nearby mountains': 'himilaya',
        }
    }

for city, city_info in cities.items():
    country = city_info['country'].title()
    population = city_info['population']
    mountains = city_info['nearby mountains'].title()

    print("\n" + city.title() + " is in " + country + ".")
    print("  It has a population of about " + str(population) + ".")
    print("  The " + mountains + " mountains are nearby.")
```

Output:

```
Santiago is in Chile.
  It has a population of about 6158080.
  The Andes mountains are nearby.

Kathmandu is in Nepal.
  It has a population of about 1003285.
  The Himilaya mountains are nearby.

Talkeetna is in Alaska.
  It has a population of about 876.
  The Alaska Range mountains are nearby.
```

7-1: Rental Car

Write a program that asks the user what kind of rental car they would like. Print a message about that car, such as “Let me see if I can find you a Subaru”.

```
car = input("What kind of car would you like? ")

print("Let me see if I can find you a " + car.title() + ".")
```

Output:

```
What kind of car would you like? Toyota Tacoma
Let me see if I can find you a Toyota Tacoma
```

7-2: Restaurant Seating

Write a program that asks the user how many people are in their dinner group. If the answer is more than eight, print a message saying they'll have to wait for a table. Otherwise, report that their table is ready.

```
party_size = input("How many people are in your dinner party tonight? ")
party_size = int(party_size)

if party_size > 8:
    print("I'm sorry, you'll have to wait for a table.")
else:
    print("Your table is ready.")
```

Output:

```
How many people are in your dinner party tonight? 12
I'm sorry, you'll have to wait for a table.
```

or:

```
How many people are in your dinner party tonight? 6
Your table is ready.
```

7-3: Multiples of Ten

Ask the user for a number, and then report whether the number is a multiple of 10 or not.

```
number = input("Give me a number, please: ")
number = int(number)

if number % 10 == 0:
    print(str(number) + " is a multiple of 10.")
else:
    print(str(number) + " is not a multiple of 10.")
```

Output:

```
Give me a number, please: 23
23 is not a multiple of 10.
```

or:

```
Give me a number, please: 90
90 is a multiple of 10.
```

7-4: Pizza Toppings

Write a loop that prompts the user to enter a series of pizza toppings until they enter a `quit` value. As they enter each topping, print a message saying you'll add that topping to their pizza.

```
prompt = "\nWhat topping would you like on your pizza?"
prompt += "\nEnter 'quit' when you are finished: "

while True:
    topping = input(prompt)
    if topping != 'quit':
        print("  I'll add " + topping + " to your pizza.")
    else:
        break
```

Output:

```
What topping would you like on your pizza?
Enter 'quit' when you are finished: pepperoni
  I'll add pepperoni to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: sausage
  I'll add sausage to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: bacon
  I'll add bacon to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: quit
```

7-5: Movie Tickets

A movie theater charges different ticket prices depending on a person's age. If a person is under the age of 3, the ticket is free; if they are between 3 and 12, the ticket is $10; and if they are over age 12, the ticket is $15. Write a loop in which

you ask users their age, and then tel them the cost of their movie ticket.

```
prompt = "How old are you?"
prompt += "\nEnter 'quit' when you are finished. "

while True:
    age = input(prompt)
    if age == 'quit':
        break
    age = int(age)

    if age < 3:
        print("  You get in free!")
    elif age < 13:
        print("  Your ticket is $10.")
    else:
        print("  Your ticket is $15.")
```

Output:

```
How old are you?
Enter 'quit' when you are finished. 2
  You get in free!
How old are you?
Enter 'quit' when you are finished. 3
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 12
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 18
  Your ticket is $15.
How old are you?
Enter 'quit' when you are finished. quit
```

7-8: Deli

Make a list called `sandwich_orders` and fill it with the names of various sandwiches. Then make an empty list called `finished_sandwiches`. Loop through the list of sandwich orders and print a message for each order, such as `I made your tuna sandwich.` As each sandwich is made, move it to the list of finished sandwiches. After all the sandwiches have been made, print a message listing each sandwich that was made.

```
sandwich_orders = ['veggie', 'grilled cheese', 'turkey', 'roast beef']
finished_sandwiches = []
```

```
while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print("I'm working on your " + current_sandwich + " sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
for sandwich in finished_sandwiches:
    print("I made a " + sandwich + " sandwich.")
```

Output:

```
I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

top

7-9: No Pastrami

Using the list `sandwich_orders` from Exercise 7-8, make sure the sandwich `'pastrami'` appears in the list at least three times. Add code near the beginning of your program to print a message saying the deli has run out of pastrami, and then use a `while` loop to remove all occurences of `'pastrami'` from `sandwich_orders`. Make sure no pastrami sandwiches end up in `finished_sandiches`.

```
sandwich_orders = [
    'pastrami', 'veggie', 'grilled cheese', 'pastrami',
    'turkey', 'roast beef', 'pastrami']
finished_sandwiches = []

print("I'm sorry, we're all out of pastrami today.")
while 'pastrami' in sandwich_orders:
    sandwich_orders.remove('pastrami')

print("\n")
while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print("I'm working on your " + current_sandwich + " sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
for sandwich in finished_sandwiches:
    print("I made a " + sandwich + " sandwich.")
```

Output:

```
I'm sorry, we're all out of pastrami today.

I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

7-10: Dream Vacation

Write a program that polls users about their dream vacation. Write a prompt similar to *If you could visit one place in the world, where would you go?* Include a block of code that prints the results of the poll.

```
name_prompt = "\nWhat's your name? "
place_prompt = "If you could visit one place in the world, where would it be? "
continue_prompt = "\nWould you like to let someone else respond? (yes/no) "

# Responses will be stored in the form {name: place}.
responses = {}

while True:
    # Ask the user where they'd like to go.
    name = input(name_prompt)
    place = input(place_prompt)

    # Store the response.
    responses[name] = place

    # Ask if there's anyone else responding.
    repeat = input(continue_prompt)
    if repeat != 'yes':
        break

# Show results of the survey.
print("\n--- Results ---")
for name, place in responses.items():
    print(name.title() + " would like to visit " + place.title() + ".")
```

Output:

```
What's your name? eric
If you could visit one place in the world, where would it be? tierra del fuego
```

```
Would you like to let someone else respond? (yes/no) yes

What's your name? erin
If you could visit one place in the world, where would it be? iceland

Would you like to let someone else respond? (yes/no) yes

What's your name? ever
If you could visit one place in the world, where would it be? death valley

Would you like to let someone else respond? (yes/no) no

--- Results ---
Ever would like to visit Death Valley.
Erin would like to visit Iceland.
Eric would like to visit Tierra Del Fuego.
```

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.

```
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.
```

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.

```
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
```

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.

```
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."
```

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.

```
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."
```

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.

```
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.
```

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.

```
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
```

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 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'}
```

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.

```
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!
```

8-9: 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
```

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.

```
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
```

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.

```
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
```

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 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!
```

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.

```
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'}
```

8-15: 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
```

9-1: Restaurant

Make a class called `Restaurant`. The `__init__()` method for `Restaurant` should store two attributes: a `restaurant_name` and a `cuisine_type`. Make a method called `describe_restaurant()` that prints these two pieces of information, and a method called `open_restaurant()` that prints a message indicating that the restaurant is open.

Make an instance called `restaurant` from your class. Print the two attributes individually, and then call both methods.

```
class Restaurant():
    """A class representing a restaurant."""
```

```
    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

restaurant = Restaurant('the mean queen', 'pizza')
print(restaurant.name)
print(restaurant.cuisine_type)

restaurant.describe_restaurant()
restaurant.open_restaurant()
```

Output:

```
The Mean Queen
pizza

The Mean Queen serves wonderful pizza.

The Mean Queen is open. Come on in!
```

top

9-2: Three Restaurants

Start with your class from Exercise 9-1. Create three different instances from the class, and call `describe_restaurant()` for each instance.

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)
```

```
    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

mean_queen = Restaurant('the mean queen', 'pizza')
mean_queen.describe_restaurant()

ludvigs = Restaurant("ludvig's bistro", 'seafood')
ludvigs.describe_restaurant()

mango_thai = Restaurant('mango thai', 'thai food')
mango_thai.describe_restaurant()
```

Output:

```
The Mean Queen serves wonderful pizza.

Ludvig'S Bistro serves wonderful seafood.

Mango Thai serves wonderful thai food.
```

top

9-3: Users

Make a class called `User`. Create two attributes called `first_name` and `last_name`, and then create several other attributes that are typically stored in a user profile. Make a method called `describe_user()` that prints a summary of the user's information. Make another method called `greet_user()` that prints a personalized greeting to the user.

Create several instances representing different users, and call both methods for each user.

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
```

```
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

willie = User('willie', 'burger', 'willieburger', 'wb@example.com', 'alaska')
willie.describe_user()
willie.greet_user()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Welcome back, e_matthes!

Willie Burger
  Username: willieburger
  Email: wb@example.com
  Location: Alaska

Welcome back, willieburger!
```

top

9-4: Number Served

Start with your program from Exercise 9-1 (page 166). Add an attribute called `number_served` with a default value of 0. Create an instance called `restaurant` from this class. Print the number of customers the restaurant has served, and then change this value and print it again.

Add a method called `set_number_served()` that lets you set the number of customers that have been served. Call this method with a new number and print the value again.

Add a method called `increment_number_served()` that lets you increment the number of customers who've been served. Call this method with any number you like that could represent how many customers were served in, say, a day of business.

```
class Restaurant():
```

```
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served


restaurant = Restaurant('the mean queen', 'pizza')
restaurant.describe_restaurant()

print("\nNumber served: " + str(restaurant.number_served))
restaurant.number_served = 430
print("Number served: " + str(restaurant.number_served))

restaurant.set_number_served(1257)
print("Number served: " + str(restaurant.number_served))

restaurant.increment_number_served(239)
print("Number served: " + str(restaurant.number_served))
```

Output:

```
The Mean Queen serves wonderful pizza.

Number served: 0
Number served: 430
Number served: 1257
Number served: 1496
```

top

9-5: Login Attempts

Add an attribute called `login_attempts` to your `User` class from Exercise 9-3 (page 166). Write amehtod called `increment_login_attempts()` that increments the value of `login_attempts` by 1. Write another method called `reset_login_attempts()` that resets the value of `login_attempts` to 0.

Make an instance of the `User` class and call `increment_login_attempts()` several times. Print the value of `login_attempts` to make sure it was incremented properly, and then call `reset_login_attempts()`. Print `login_attempts` again to make sure it was reset to 0.

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

print("\nMaking 3 login attempts...")
eric.increment_login_attempts()
eric.increment_login_attempts()
eric.increment_login_attempts()
print("  Login attempts: " + str(eric.login_attempts))

print("Resetting login attempts...")
eric.reset_login_attempts()
```

```
print("  Login attempts: " + str(eric.login_attempts))
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Welcome back, e_matthes!

Making 3 login attempts...
  Login attempts: 3
Resetting login attempts...
  Login attempts: 0
```

top

9-6: Ice Cream Stand

An ice cream stand is a specific kind of restaurant. Write a class called `IceCreamStand` that inherits from the `Restaurant` class you wrote in Exercise 9-1 (page 166) or Exercise 9-4 (page 171). Eitehr version of the class will work; just pick the one you like better. Add an attribute called `flavors` that stores a list of ice cream flavors. Write a method that displays theese flavors. Create an instance of `IceCreamStand`, and call this method.

```
class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
```

```
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served


class IceCreamStand(Restaurant):
    """Represent an ice cream stand."""

    def __init__(self, name, cuisine_type='ice_cream'):
        """Initialize an ice cream stand."""
        super().__init__(name, cuisine_type)
        self.flavors = []

    def show_flavors(self):
        """Display the flavors available."""
        print("\nWe have the following flavors available:")
        for flavor in self.flavors:
            print("- " + flavor.title())


big_one = IceCreamStand('The Big One')
big_one.flavors = ['vanilla', 'chocolate', 'black cherry']

big_one.describe_restaurant()
big_one.show_flavors()
```

Output:

```
The Big One serves wonderful ice_cream.

We have the following flavors available:
- Vanilla
- Chocolate
- Black Cherry
```

top

9-7: Admin

An administrator is a special kind of user. Write a class called `Admin` that inherits from the `User` class you wrote in Exercise 9-3 (page 166) or Exercise 9-5 (page 171). Add an attribute, `privileges`, that stores a list of strings like `"can add post"`, `"can delete post"`, `"can ban user"`, and so on. WRite a method called `show_privileges()` that lists the administrator's set of privileges. Create an instance of `Admin`, and call your method.

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
```

```
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)
        self.privileges = []

    def show_privileges(self):
        """Display the privileges this administrator has."""
        print("\nPrivileges:")
        for privilege in self.privileges:
            print("- " + privilege)


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]

eric.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

top

9-8: Privileges

Write a separate `Privileges` class. The class should have one attribute, `privileges`, that stores a list of strings as described in Exercise 9-7. Move the `show_privileges()` method to this class. Make a `Privileges` instance as an attribute in the `Admin` class. Create a new instance of `Admin` and use your method to show its privileges.

```
class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0
```

```
class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges()

class Privileges():
    """A class to store an admin's privileges."""

    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print("- " + privilege)
        else:
            print("- This user has no privileges.")


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges.show_privileges()

print("\nAdding privileges...")
eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- This user has no privileges.

Adding privileges...

Privileges:
- can reset passwords
```

```
- can moderate discussions
- can suspend accounts
```

top

9-9: Battery Upgrade

Use the final version of *electric_car.py* from this section. Add a method to the `Battery` class called `upgrade_battery()`. This method should check the battery size and set the capacity to 85 if it isn't already. Make an electric car with a default battery size, call `get_range()` once, and then call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

```
class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles

class Battery():
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=60):
        """Initialize the batteery's attributes."""
        self.battery_size = battery_size
```

```
    def describe_battery(self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")


    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185

        message = "This car can go approximately " + str(range)
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        """Upgrade the battery if possible."""
        if self.battery_size == 60:
            self.battery_size = 85
            print("Upgraded the battery to 85 kWh.")
        else:
            print("The battery is already upgraded.")


class ElectricCar(Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__(self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()


print("Make an electric car, and check the battery:")
my_tesla = ElectricCar('tesla', 'model s', 2016)
my_tesla.battery.describe_battery()

print("\nUpgrade the battery, and check it again:")
my_tesla.battery.upgrade_battery()
my_tesla.battery.describe_battery()

print("\nTry upgrading the battery a second time.")
my_tesla.battery.upgrade_battery()
my_tesla.battery.describe_battery()
```

Output:

```
Make an electric car, and check the battery:
This car has a 60-kWh battery.
```

```
Upgrade the battery, and check it again:
Upgraded the battery to 85 kWh.
This car has a 85-kWh battery.

Try upgrading the battery a second time.
The battery is already upgraded.
This car has a 85-kWh battery.
```

top

9-10: Imported Restaurant

Using your latest `Restaurant` class, store it in a module. Make a separate file that imports `Restaurant`. Make a `Restaurant` instance, and call one of `Restaurant`'s methods to show that the `import` statement is working properly.

restaurant.py:

```
"""A class representing a restaurant."""

class Restaurant():
    """A class representing a restaurant."""

    def __init__(self, name, cuisine_type):
        """Initialize the restaurant."""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """Display a summary of the restaurant."""
        msg = self.name + " serves wonderful " + self.cuisine_type + "."
        print("\n" + msg)

    def open_restaurant(self):
        """Display a message that the restaurant is open."""
        msg = self.name + " is open. Come on in!"
        print("\n" + msg)

    def set_number_served(self, number_served):
        """Allow user to set the number of customers that have been served."""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """Allow user to increment the number of customers served."""
        self.number_served += additional_served
```

my_restaurant.py:

```
from restaurant import Restaurant
```

```
channel_club = Restaurant('the channel club', 'steak and seafood')
channel_club.describe_restaurant()
channel_club.open_restaurant()
```

Output:

```
The Channel Club serves wonderful steak and seafood.

The Channel Club is open. Come on in!
```

top

9-11: Imported Admin

Start with your work from Exercise 9-8 (page 178). Store the classes `User`, `Privileges` and `Admin` in one module. Create a separate file, make an `Admin` instance, and call `show_priveleges()` to show that everything is working correctly.

user.py:

```
"""A collection of classes for modeling users."""

class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1
```

```
    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0


class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges([])


class Privileges():
    """Stores privileges associated with an Admin account."""

    def __init__(self, privileges):
        """Initialize the privileges object."""
        self.privilege = privileges

    def show_privileges(self):
        """Display the privileges this administrator has."""
        for privilege in self.privileges:
            print("- " + privilege)
```

my_user.py:

```
from user import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print("\nThe admin " + eric.username + " has these privileges: ")
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
```

```
- can moderate discussions
- can suspend accounts
```

top

9-12: Multiple Modules

Store the `User` class in one module, and store the `Privileges` and `Admin` classes in a separate module. In a separate file, create an `Admin` instance and call `show_privileges()` to show that everything is still working correctly.

user.py:

```
"""A class for modeling users."""

class User():
    """Represent a simple user profile."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the user."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """Display a summary of the user's information."""
        print("\n" + self.first_name + " " + self.last_name)
        print("  Username: " + self.username)
        print("  Email: " + self.email)
        print("  Location: " + self.location)

    def greet_user(self):
        """Display a personalized greeting to the user."""
        print("\nWelcome back, " + self.username + "!")

    def increment_login_attempts(self):
        """Increment the value of login_attempts."""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """Reset login_attempts to 0."""
        self.login_attempts = 0
```

admin.py:

```
"""A collection of classes for modeling an admin user account."""

from user import User
```

```
class Admin(User):
    """A user with administrative privileges."""

    def __init__(self, first_name, last_name, username, email, location):
        """Initialize the admin."""
        super().__init__(first_name, last_name, username, email, location)

        # Initialize an empty set of privileges.
        self.privileges = Privileges([])


class Privileges():
    """Stores privileges associated with an Admin account."""

    def __init__(self, privileges):
        """Initialize the privileges object."""
        self.privilege = privileges

    def show_privileges(self):
        """Display the privileges this administrator has."""
        for privilege in self.privileges:
            print("- " + privilege)
```

my_admin.py

```
from admin import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print("\nThe admin " + eric.username + " has these privileges: ")
eric.privileges.show_privileges()
```

Output:

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

9-13: OrderedDict Rewrite

Start with Exercise 6-4 (page 108), where you used a standard dictionary to represent a glossary. Rewrite the program using the `OrderedDict` class and make sure the order of the output matches the order in which key-value pairs were added to the dictionary.

Note: *In Python 3.6, dictionaries store keys in order. However, this is not guaranteed to work in all versions of Python, so you should still use an OrderedDict when you need key-value pairs to be stored in a particular order.*

```
from collections import OrderedDict

glossary = OrderedDict()

glossary['string'] = 'A series of characters.'
glossary['comment'] = 'A note in a program that the Python interpreter ignores.'
glossary['list'] = 'A collection of items in a particular order.'
glossary['loop'] = 'Work through a collection of items, one at a time.'
glossary['dictionary'] = "A collection of key-value pairs."
glossary['key'] = 'The first item in a key-value pair in a dictionary.'
glossary['value'] = 'An item associated with a key in a dictionary.'
glossary['conditional test'] = 'A comparison between two values.'
glossary['float'] = 'A numerical value with a decimal component.'
glossary['boolean expression'] = 'An expression that evaluates to True or False.'

for word, definition in glossary.items():
    print("\n" + word.title() + ": " + definition)
```

Output:

```
String: A series of characters.

Comment: A note in a program that the Python interpreter ignores.

List: A collection of items in a particular order.

Loop: Work through a collection of items, one at a time.

Dictionary: A collection of key-value pairs.

Key: The first item in a key-value pair in a dictionary.

Value: An item associated with a key in a dictionary.

Conditional Test: A comparison between two values.

Float: A numerical value with a decimal component.

Boolean Expression: An expression that evaluates to True or False.
```

9-14: Dice

The module `random` contains functions that generate random numbers in a variety of ways. The function `randint()` returns an integer in the range you provide. the following code returns a number between 1 and 6:

```
from random import randint
x = randint(1, 6)
```

Make a class `Die` with one attribute called `sides`, which has a default value of 6. Write a method called `roll_die()` that prints a random number between 1 and the number of sides the die has. Make a 6-sided die and roll it 10 times.

Make a 10-sided die and a 20-sided die. Roll each die 10 times.

```
from random import randint

class Die():
    """Represent a die, which can be rolled."""

    def __init__(self, sides=6):
        """Initialize the die."""
        self.sides = sides

    def roll_die(self):
        """Return a number between 1 and the number of sides."""
        return randint(1, self.sides)

# Make a 6-sided die, and show the results of 10 rolls.
d6 = Die()

results = []
for roll_num in range(10):
    result = d6.roll_die()
    results.append(result)
print("10 rolls of a 6-sided die:")
print(results)

# Make a 10-sided die, and show the results of 10 rolls.
d10 = Die(sides=10)

results = []
for roll_num in range(10):
    result = d10.roll_die()
    results.append(result)
print("\n10 rolls of a 10-sided die:")
print(results)

# Make a 20-sided die, and show the results of 10 rolls.
d20 = Die(sides=20)
```

```
results = []
for roll_num in range(10):
    result = d20.roll_die()
    results.append(result)
print("\n10 rolls of a 20-sided die:")
print(results)
```

Output:

```
10 rolls of a 6-sided die:
[5, 5, 6, 3, 6, 4, 2, 2, 1, 1]

10 rolls of a 10-sided die:
[8, 9, 8, 10, 7, 1, 3, 5, 3, 4]

10 rolls of a 20-sided die:
[4, 3, 18, 17, 3, 1, 13, 12, 5, 14
```