W06 Learning Activity: Objects

A *paradigm* is a way of thinking or a way of perceiving the world. There are at least four main paradigms for programming a computer: procedural, declarative, functional, and object-oriented. During most of CSE 110 and 111, you used procedural programming. During this week, you will be introduced to object-oriented programming.

Programming Paradigms

Procedural Programming

Procedural programming is a way of programming that focuses on the process or the steps to accomplish a task. For example, if we had 100 numbers and wanted to know the average value of those 100 numbers, we could add the numbers and then divide by 100. This is one process to compute the average of numbers: add them and divide by the quantity of numbers. A Python procedural program for computing the average is shown in example 1.

```
# Example 1
def main():
    numbers = [87, 95, 72, 92, 95, 88, 84]
    total = 0
    for x in numbers:
        total += x
    average = total / len(numbers)
    print(f"average: {average:.2f}")
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_1.py
average: 87.57
```

Notice that with procedural programming, we must write the process or the steps that are necessary to complete a task. Procedural programming is the type of

programming that you did most often in CSE 110 and 111.

Declarative Programming

When we use declarative programming to program a computer, we do not focus on the process or steps to accomplish a task, but rather we focus on what we want from the task, or in other words, we focus on the desired result. Continuing the example of the average, with declarative programming, we focus on exactly what numbers we want averaged and tell the computer to compute that average for us. SQL is a declarative programming language used with relational databases. Example 2 contains SQL code that causes the computer to compute the average of a column of numbers.

```sql
-- Example 2
SELECT AVG(numbers) FROM table;
```

```
   AVG(numbers)
----------------
87.57142857142857
```

Notice in example 2, that the code does not contain the steps required to compute the average. Someone else already wrote the code that contains those steps. Instead, the SQL code contains a command that tells the computer to compute the average of a column named *numbers*. The term “declarative programming” means that we write or declare what we want the computer to do. We do not tell the computer how to compute something. We declare what we want the computer to do, and the computer determines how to do it and then does it.

Functional Programming

When we use functional programming to program a computer, we focus on the functions necessary to accomplish a task. Mathematicians often find functional programming natural for them because they are accustomed to using functions while studying mathematics. In functional programming, functions are so important that we often pass functions into other functions.

```python
# Example 3
from functools import reduce
def main():
    numbers = [87, 95, 72, 92, 95, 88, 84]
```

```
    func_add = lambda a, b: a + b
    total = reduce(func_add, numbers)
    average =  total / len(numbers)
    print(f"average: {average:.2f}")
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_3.py
average: 87.57
```

Notice how example 3 uses three functions: a lambda function, the **`reduce`** function, and the **`len`** function. Notice also that the lambda function is passed into the **`reduce`** function. Passing a function into a function is one of the marks of functional programming.

Object-Oriented Programming

Object-oriented programming is a programming paradigm based on the concept of objects. An *object* is a piece of a program that contains both data (also known as attributes) and functions (also known as methods).

When we write an object-oriented program, we combine data and functions together into objects. For example, if we were writing a registration program used by students to register for courses at a university, we would write code to create **`Student`** objects and **`Course`** objects. Each **`Student`** object would have data such as *given_name*, *family_name*, and *phone_number* and would have functions such as **`register`**, **`enroll`**, **`drop`**, and **`withdraw`**. Each **`Course`** object would have data such as *course_code*, *title*, *description*, and *list_of_students* and would have functions such as **`get_students`** and **`take_role`**.

Python includes many built-in and standard objects that a programmer can use to write programs. In fact, you have already used many objects in your programs. Python lists and dictionaries are objects and have attributes and methods. Readers and Writers from the **`csv`** module are also objects.

One of the marks of object-oriented programming is selecting attributes and calling methods using the *dot operator* (a period). The official name of the dot operator is *component selector*, but almost no one calls it that because the term “dot” is much easier to say than “component selector.” The code in example 4 uses the dot operator (.) to call the **`append`** method.

```
1 # Example 4
2 def main():
3     numbers = [87, 95, 72, 92, 95, 88, 84]
4     numbers.append(78)
5     numbers.append(72)
6     print(numbers)
7 # Call main to start this program.
8 if __name__ == "__main__":
9     main()
```

```
> python example_4.py
[87, 95, 72, 92, 95, 88, 84, 78, 72]
```

There are several types of commands that are commonly found in object-oriented programs. These types of commands are so common, that a programmer must be able to recognize and write them. Three of these types of commands are:

1. Creating objects, for example:

```
obj = datetime.now()
```

2. Accessing the attributes of an object using the dot operator (.), for example:

```
year = obj.year
```

3. Calling the methods of an object using the dot operator (.), for example:

```
new_obj = obj.replace(year=2035)
day_of_week = obj.weekday()
```

Python Lists Are Objects

In Python, lists are objects with attributes and methods, and a programmer can modify a list by calling those methods. The list methods are documented in a section of the Python Tutorial titled More on Lists.

Example 5 below contains a program that is similar to example 2 in the preparation content 1 of Week 4. Now that you know what an object is, that objects have methods, and that Python lists are objects, this example code should make more sense than it did in week 4. Notice that the **append** method is called on lines 6–8, **insert** is called on line 10, **index** is called on line 13, **pop** is called on line 18, and **remove** is called on line 20.

```python
# Example 5
def main():
    # Create an empty list that will hold fabric names.
    fabrics = []
    # Add three elements at the end of the fabrics list.
    fabrics.append("velvet")
    fabrics.append("denim")
    fabrics.append("gingham")
    # Insert an element at the beginning of the fabrics list.
    fabrics.insert(0, "chiffon")
    print(fabrics)
    # Get the index where velvet is stored in the fabrics list.
    i = fabrics.index("velvet")
    # Replace velvet with taffeta.
    fabrics[i] = "taffeta"
    print(fabrics)
    # Remove the last element from the fabrics list.
    fabrics.pop()
    # Remove denim from the fabrics list.
    fabrics.remove("denim")
    print(fabrics)
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_5.py
['chiffon', 'velvet', 'denim', 'gingham']
['chiffon', 'taffeta', 'denim', 'gingham']
['chiffon', 'taffeta']
```

Python Dictionaries Are Objects

Python dictionaries are objects with attributes and methods, and a programmer can modify a dictionary by calling those methods. There doesn't seem to be an official Python web page that documents the dictionary methods, so here is a list of the built-in dictionary methods:

Method	Description
d.clear()	Removes all the elements from the dictionary *d*.
d.copy()	Returns a copy of the dictionary *d*.

Method	Description
d.get(*key*)	Returns the value of the specified **key**. Calling the get method is almost equivalent to using square brackets ([and]) to find a key in a dictionary.
d.items()	Returns a list that contains the key value pairs that are in the dictionary *d*.
d.keys()	Returns a list that contains the keys that are in the dictionary *d*.
d.pop(*key*)	Removes the element with the specified *key* from the dictionary *d*.
d.update(*other*)	Updates the dictionary *d* with the key value pairs that are in the *other* dictionary.
d.values()	Returns a list that contains the values that are in the dictionary *d*.

The following example code, which is similar to example 1 from the preparation content 2 of week 4, calls dictionary methods at lines 17, 23, and 31.

```
1  # Example 6
2  def main():
3      # Create a dictionary with student IDs as
4      # the keys and student names as the values.
5      students = {
6          "42-039-4736": "Clint Huish",
7          "61-315-0160": "Amelia Davis",
8          "10-450-1203": "Ana Soares",
9          "75-421-2310": "Abdul Ali",
10         "07-103-5621": "Amelia Davis",
11         "81-298-9238": "Sama Patel"
12     }
13     # Get a student ID from the user.
14     id = input("Enter a student ID: ")
15     # Lookup the student ID in the dictionary and
16     # retrieve the corresponding student name.
17     name = students.get(id)
18     if name:
19         # Print the student name.
20         print(name)
```

```
        # Remove the student that the user
        # specified from the dictionary.
        students.pop(id)
    else:
        print("No such student")
    print()
    # Use a for loop to print each key value pair
    # in the dictionary. Of course, the code in
    # the body of a loop can do much more with
    # each key value pair than simply print it.
    for key, value in students.items():
        print(key, value)
# Call main to start this program.
if __name__ == "__main__":
    main()
```

```
> python example_6.py
Enter a student ID: 81-298-9238
Sama Patel
42-039-4736 Clint Huish
61-315-0160 Amelia Davis
10-450-1203 Ana Soares
75-421-2310 Abdul Ali
07-103-5621 Amelia Davis
```

Summary

This lesson introduces you to object-oriented programming. You are learning that an object has data (attributes) and functions (methods) and that a programmer uses the dot operator (.) to access the attributes and call the methods in an object. Python lists and dictionaries are objects and contain attributes and methods.

W06 Checkpoint: Using Objects

Purpose

Improve your ability to write object-oriented code.

Problem Statement

There are several types of commands that are commonly found in object oriented programs. These types of commands are so common, that a programmer must be

able to recognize and write them. One of these types of commands is calling the methods of an object using the dot operator (.) as shown in this template:

```
variable = object.method(arg1, arg2, ...)
```

Helpful Documentation

- In Python, lists are objects with attributes and methods, and a programmer can modify a list by calling those methods. The list methods are documented in a web page titled More on Lists.

Assignment

Write a small Python program named **`fruit.py`** that demonstrates object oriented programming by modifying a list. Do the following:

1. Open a new blank file in VS Code and save it as **`fruit.py`**
2. Copy and paste this code at the top of your **`fruit`** program:

```
def main():
    # Create and print a list named fruit.
    fruit_list = ["pear", "banana", "apple", "mango"]
    print(f"original: {fruit_list}")
```

3. Add code to reverse and print *fruit_list*.
4. Add code to append "orange" to the end of *fruit_list* and print the list.
5. Add code to find where "apple" is located in *fruit_list* and insert "cherry" before "apple" in the list and print the list.
6. Add code to remove "banana" from *fruit_list* and print the list.
7. Add code to pop the last element from *fruit_list* and print the popped element and the list.
8. Add code to sort and print *fruit_list*.
9. Add code to clear and print *fruit_list*.
10. At the bottom of your program write a call to the **`main`** function.

Testing Procedure

Verify that your program works correctly by following each step in this testing procedure:

1. Run your program and ensure that your program's output is the same as the output shown below.

```
> python fruit.py
original: ['pear', 'banana', 'apple', 'mango']
reversed: ['mango', 'apple', 'banana', 'pear']
append orange: ['mango', 'apple', 'banana', 'pear', 'orange']
insert cherry: ['mango', 'cherry', 'apple', 'banana', 'pear', 'orange']
remove banana: ['mango', 'cherry', 'apple', 'pear', 'orange']
pop orange: ['mango', 'cherry', 'apple', 'pear']
sorted: ['apple', 'cherry', 'mango', 'pear']
cleared: []
```

Ponder

After you finish your program, examine the code in **`main`** and realize that you wrote object oriented code when you used the **`fruit_list`** and the dot operator (.) to call the **`reverse`**, **`append`**, **`insert`**, **`remove`**, **`pop`**, **`sort`**, and **`clear`** methods.

Sample Solution

When your program is finished, view the sample solution for this assignment to compare your solution to that one. Before looking at the sample solution, you should work to complete this checkpoint program. However, if you have worked on it for at least an hour and are still having problems, feel free to use the sample solution to help you finish your program.

Submission

When complete, report your progress in the associated Canvas quiz.

Up Next

- W06 Check Your Understanding Quiz

Useful Links:

- Return to: Week Overview | Course Home