BYU IDAHO CSE 111 | Programming with Functions ☼

12 Prepare: Using Objects

A paradigm is a way of thinking or a way of viewing the world. There are at least four main paradigms for programming a computer: procedural, declarative, functional, and object-oriented. During CSE 110 and 111, you used or encountered the first three of these programming paradigms. During this lesson, you will be introduced to the last: 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.

```python
# 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(average)
```

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;
```

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. You did this in the checkpoint and team activity for lesson 11. Example 3 contains a functional programming solution to computing the average in 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(average)
```

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 school, 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.

```
# Example 4

def main():
    numbers = [87, 95, 72, 92, 95, 88, 84]
    numbers.append(78)
    numbers.append(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=2025)
```

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 contains a program copied from the prepare content of lesson 7. 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 lesson 7. Notice that the append method is called on lines 8–10, insert is called on line 13, index is called on line 16, pop is called on line 22, and remove is called on line 25.

```
1    # Example 5
2
3    def main():
4        # Create an empty list that will hold fabric names.
5        fabrics = []
6
7        # Add three elements at the end of the fabrics list.
8        fabrics.append("velvet")
9        fabrics.append("denim")
10       fabrics.append("gingham")
11
12       # Insert an element at the beginning of the fabrics list.
13       fabrics.insert(0, "chiffon")
14
15       # Get the index where velvet is stored in the fabrics list.
16       i = fabrics.index("velvet")
17
18       # Replace velvet with taffeta.
19       fabrics[i] = "taffeta"
20
21       # Remove the last element from the fabrics list.
22       fabrics.pop()
23
24       # Remove denim from the fabrics list.
25       fabrics.remove("denim")
26
27
28   # Call main to start this program.
29   if __name__ == "__main__":
30       main()
```

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*.
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.

Method	Description
d.items()	Returns a list of the key value pairs 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 of all the values in the dictionary *d*.

The following example code, which is from the prepare content of lesson 3, calls dictionary methods at lines 20, 28, and 36.

```python
 1   def main():
 2       # Create a dictionary with student IDs as
 3       # the keys and student names as the values.
 4       students = {
 5           "42-039-4736": "Clint Huish",
 6           "61-315-0160": "Michelle Davis",
 7           "10-450-1203": "Jorge Soares",
 8           "75-421-2310": "Abdul Ali",
 9           "07-103-5621": "Michelle Davis"
10       }
11
12       # Add an item to the dictionary.
13       students["81-298-9238"] = "Sama Patel"
14
15       # Get a student ID from the user.
16       id = input("Enter a student ID: ")
17
18       # Lookup the student ID in the dictionary and
19       # retrieve the corresponding student name.
20       name = students.get(id)
21
22       if name:
23           # Print the student name.
24           print(name)
25
26           # Remove the student that the user
27           # specified from the dictionary.
28           students.pop(id)
29       else:
30           print("No such student")
31
32       # Use a for loop to print each key value pair in the
33       # dictionary. Of course, the code in the body of a loop
34       # can do much more with each key value pair than simply
35       # print it.
36       for key, value in students.items():
37           print(key, value)
38
39
40   # Call main to start this program.
41   if __name__ == "__main__":
42       main()
```

Summary

This lesson is introducing 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.