

# Object Oriented Programming II

Dr. Burak Kaleci

March 1, 2019

# Content

Procedural and Object-Oriented Programming

Classes

# Procedural Programming

- ▶ There are primarily two methods of programming in use today: procedural and object oriented.
- ▶ The earliest programming languages were procedural, meaning a program was made of one or more procedures.
- ▶ You can think of a procedure **simply as a function** that performs a specific task such as gathering input from the user, performing calculations, reading or writing files, displaying output, and so on.
- ▶ The programs that you have written so far have been procedural in nature.

# Procedural Programming

- ▶ Typically, procedures operate on data items that are separate from the procedures.
- ▶ In a procedural program, the data items are commonly passed from one procedure to another.
- ▶ As you might imagine, the focus of procedural programming is on the creation of procedures that operate on the program's data.
- ▶ The separation of data and the code that operates on the data can lead to problems, however, as the program becomes larger and more complex.

# Procedural Programming

- ▶ For example, suppose you are part of a programming team that has written an extensive customer database program.
- ▶ The program was initially designed so a customer's name, address, and phone number were referenced by three variables.
- ▶ Your job was to design several functions that accept those three variables as arguments and perform operations on them.
- ▶ The software has been operating successfully for some time, but your team has been asked to update it by adding several new features.

# Procedural Programming

- ▶ During the revision process, the senior programmer informs you that the customer's name, address, and phone number will no longer be stored in variables.
- ▶ Instead, they will be stored in a list.
- ▶ This means you will have to modify all of the functions that you have designed so they accept and work with a list instead of the three variables.
- ▶ Making these extensive modifications not only is a great deal of work, but also opens the opportunity for errors to appear in your code.

# Object-Oriented Programming

- ▶ Whereas procedural programming is centered on creating procedures (functions), object-oriented programming (OOP) is centered on creating objects.
- ▶ **An object is a software entity that contains both data and procedures.**
- ▶ The data contained in an object is known as the **object's data attributes**.
- ▶ An **object's data attributes** are simply **variables** that reference data.

# Object-Oriented Programming

- ▶ The **procedures** that an object performs are known as **methods**.
- ▶ An object's methods are functions that perform operations on the object's data attributes.
- ▶ The object is, conceptually, a self-contained unit that consists of data attributes and methods that operate on the data attributes.





# Object-Oriented Programming

- ▶ OOP addresses the problem of code and data separation through **encapsulation** and **data hiding**.
- ▶ **Encapsulation refers to the combining of data and code into a single object.**
- ▶ **Data hiding refers to an object's ability to hide its data attributes from code that is outside the object.**
- ▶ **Only the object's methods may directly access and make changes to the object's data attributes.**
- ▶ **An object typically hides its data, but allows outside code to access its methods.**
- ▶ The object's methods provide programming statements outside the object with indirect access to the object's data attributes.

# Object-Oriented Programming

- ▶ When an object's data attributes are hidden from outside code, and access to the data attributes is restricted to the object's methods, the data attributes are protected from accidental corruption.
- ▶ **In addition, the code outside the object does not need to know about the format or internal structure of the object's data.**
- ▶ The code only needs to interact with the object's methods.
- ▶ When a programmer changes the structure of an object's internal data attributes, he or she also modifies the object's methods so they may properly operate on the data.
- ▶ **The way in which outside code interacts with the methods, however, does not change.**

# The Automobile as an Object

- ▶ To help you understand objects and their contents, let's begin with a simple analogy. Suppose you want to drive a car and make it go faster by pressing its accelerator pedal.
- ▶ What must happen before you can do this?
- ▶ Well, before you can drive a car, some one has to design it.
- ▶ A car typically begins as engineering drawings, similar to the blueprints that describe the design of a house.
- ▶ These drawings include the design for an accelerator pedal. The pedal hides from the driver the complex mechanisms that actually make the car go faster, just as the brake pedal hides the mechanisms that slow the car, and the steering wheel "hides" the mechanisms that turn the car.

# The Automobile as an Object

- ▶ This enables people with little or no knowledge of how engines,braking and steering mechanisms work to drive a car easily, simply by using the accelerator pedal, the break pedal, the steering wheel, the transmission mechanism and other such simple and user-friendly "interfaces" to the car's complex internal mechanism.
- ▶ You cannot drive a car's engineering drawings.
- ▶ Before you can drive a car, it must be built from the engineering drawings that describe it.
- ▶ A completed car has an actual accelerator pedal to make the car go faster.
- ▶ But even that's not enough. The car won't accelerate on its own,so the driver must press the pedal to accelerate the car to tell the car to go faster.

# Methods and Classes

- ▶ Let's use our car example to introduce some key object-oriented programming concepts.
- ▶ Performing a task in a program requires a method, which houses the program statements that actually perform its task.
- ▶ The method hides these statements from its user, just as the accelerator pedal of a car hides from the driver the mechanisms of making the car go faster.
- ▶ In Python, we create a program unit called a class to house the set of methods that perform the class's tasks.
- ▶ A class is similar in concept to a car's engineering drawings, which house the design of an accelerator pedal, steering wheel, and so on.

# Instantiation

- ▶ Just as some one has to build a car from its engineering drawings before you can actually drive a car, you must build an object of a class before a program can perform the tasks that the class's methods define.
- ▶ The process of doing this is called instantiation.
- ▶ An object is then referred to as an instance of its class.
- ▶ Note also that just as many cars can be built from the same engineering drawing, many objects can be built from the same class.

# Messages and Method Calls

- ▶ When you drive a car, pressing its gas pedal sends a message to the car to perform a task that is to go faster.
- ▶ Similarly, you send messages to an object. Each message is implemented as a method call that tells a method of the object to perform its task.

# Attributes and Data

- ▶ A car, besides having capabilities to accomplish tasks, also has attributes, such as its color, its number of doors, the amount of gas in its tank, its current speed and its record of total miles driven.
- ▶ Like its capabilities, the car's attributes are represented as part of its design in its engineering diagrams.
- ▶ As you drive an actual car, these attributes are carried along with the car. Every car maintains its own attributes. For example, each car knows how much gas is in its own gas tank, but not how much is in the tanks of other cars.
- ▶ An object, similarly, has attributes that it carries along as it's used in a program. These attributes are specified as part of the object's class.
- ▶ Attributes are specified by the class's variables.



# Object Reusability

- ▶ In addition to solving the problems of code and data separation, the use of OOP has also been encouraged by the trend of **object reusability**.
- ▶ An object is not a stand-alone program, but is used by programs that need its services.
- ▶ For example, Sharon is a programmer who has developed a set of objects for rendering 3D images.
- ▶ She is a math whiz and knows a lot about computer graphics, so her objects are coded to perform all of the necessary 3D mathematical operations and handle the computer's video hardware.
- ▶ Tom, who is writing a program for an architectural firm, needs his application to display 3D images of buildings.
- ▶ Because he is working under a tight deadline and does not possess a great deal of knowledge about computer graphics, he can use Sharon's objects to perform the 3D rendering.

# An Everyday Example of an Object

- ▶ Imagine that your alarm clock is actually a software object. If it were, it would have the following data attributes:
  - current\_second (a value in the range of 0-59)
  - current\_minute (a value in the range of 0-59)
  - current\_hour (a value in the range of 1-12)
  - alarm\_time (a valid hour and minute)
  - alarm\_is\_set (True or False)
- ▶ As you can see, the data attributes are merely values that define the state in which the alarm clock is currently.
- ▶ You, the user of the alarm clock object, cannot directly manipulate these data attributes because they are **private**.

# An Everyday Example of an Object

- ▶ To change a data attribute's value, you must use one of the object's methods.
- ▶ The following are some of the alarm clock object's methods:
  - set\_time
  - set\_alarm\_time
  - set\_alarm\_on
  - set\_alarm\_off
- ▶ Each method manipulates one or more of the data attributes.
- ▶ For example, the set\_time method allows you to set the alarm clock's time. You activate the method by pressing a button on top of the clock.
- ▶ By using another button, you can activate the set\_alarm\_time method.
- ▶ In addition, another button allows you to execute the set\_alarm\_on and set\_alarm\_off methods.
- ▶ **Notice all of these methods can be activated by you, who are outside the alarm clock.**
- ▶ Methods that can be accessed by entities outside the object are known as **public methods**.

# An Everyday Example of an Object

- ▶ The alarm clock also has **private methods**, which are part of the object's private, internal workings.
- ▶ **External entities (such as you, the user of the alarm clock) do not have direct access to the alarm clock's private methods.**
- ▶ The object is designed to execute these methods automatically and hide the details from you.
- ▶ The following are the alarm clock object's private methods:
  - increment\_current\_second
  - increment\_current\_minute
  - increment\_current\_hour
  - sound\_alarm

# An Everyday Example of an Object

- ▶ Every second the `increment_current_second` method executes. This changes the value of the `current_second` data attribute.
- ▶ If the `current_second` data attribute is set to 59 when this method executes, the method is programmed to reset `current_second` to 0, and then cause the `increment_current_minute` method to execute.
- ▶ The `increment_current_minute` method adds 1 to the `current_minute` data attribute, unless it is set to 59. In that case, it resets `current_minute` to 0 and causes the `increment_current_hour` method to execute.
- ▶ The `increment_current_minute` method compares the new time to the `alarm_time`. If the two times match and the alarm is turned on, the `sound_alarm` method is executed.

# Introduction

- ▶ Now, let's discuss how objects are created in software. Before an object can be created, it must be designed by a programmer.
- ▶ The programmer determines the data attributes and methods that are necessary, then creates a class.
- ▶ **A class is code that specifies the data attributes and methods of a particular type of object.**
- ▶ **Think of a class as a "blueprint" from which objects may be created. It serves a similar purpose as the blueprint for a house.**
- ▶ **The blueprint itself is not a house, but is a detailed description of a house.** When we use the blueprint to build an actual house, we could say we are building an instance of the house described by the blueprint.
- ▶ If we so desire, we can build several identical houses from the same blueprint. Each house is a separate instance of the house described by the blueprint.

# Class Definitions

- ▶ To create a class, you write a class definition.
- ▶ **A class definition is a set of statements that define a class's methods and data attributes.**
- ▶ Let's look at a simple example. Suppose we are writing a program to simulate the tossing of a coin.
- ▶ In the program, we need to repeatedly toss the coin and each time determine whether it landed heads up or tails up.
- ▶ Taking an object-oriented approach, we will write a class named Coin that can perform the behaviors of the coin.

# Class Definitions

```
1 import random
2
3 # The Coin class simulates a coin that can
4 # be flipped.
5
6 class Coin:
7
8     # The __init__ method initializes the
9     # sideup data attribute with 'Heads'.
10
11     def __init__(self):
12         self.sideup = 'Heads'
13
14     # The toss method generates a random number
15     # in the range of 0 through 1. If the number
16     # is 0, then sideup is set to 'Heads'.
17     # Otherwise, sideup is set to 'Tails'.
18
19     def toss(self):
20         if random.randint(0, 1) == 0:
21             self.sideup = 'Heads'
22         else:
23             self.sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.sideup
```

- ▶ In line 1, we import the random module. This is necessary because we use the randint function to generate a random number.
- ▶ Line 6 is the beginning of the class definition.
- ▶ **It begins with the keyword class, followed by the class name, which is Coin, followed by a colon.**



# Class Definitions

```
1 import random
2
3 # The Coin class simulates a coin that can
4 # be flipped.
5
6 class Coin:
7
8     # The __init__ method initializes the
9     # sideup data attribute with 'Heads'.
10
11     def __init__(self):
12         self.sideup = 'Heads'
13
14     # The toss method generates a random number
15     # in the range of 0 through 1. If the number
16     # is 0, then sideup is set to 'Heads'.
17     # Otherwise, sideup is set to 'Tails'.
18
19     def toss(self):
20         if random.randint(0, 1) == 0:
21             self.sideup = 'Heads'
22         else:
23             self.sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.sideup
```

- ▶ The same rules that apply to variable names also apply to class names.
- ▶ **However, notice that we started the class name, Coin, with an uppercase letter.**
- ▶ **This is not a requirement, but it is a widely used convention among programmers.**
- ▶ This helps to easily distinguish class names from variable names when reading code.

# Class Definitions

```
1 import random
2
3 # The Coin class simulates a coin that can
4 # be flipped.
5
6 class Coin:
7
8     # The __init__ method initializes the
9     # sideup data attribute with 'Heads'.
10
11     def __init__(self):
12         self.sideup = 'Heads'
13
14     # The toss method generates a random number
15     # in the range of 0 through 1. If the number
16     # is 0, then sideup is set to 'Heads'.
17     # Otherwise, sideup is set to 'Tails'.
18
19     def toss(self):
20         if random.randint(0, 1) == 0:
21             self.sideup = 'Heads'
22         else:
23             self.sideup = 'Tails'
24
25     # The get_sideup method returns the value
26     # referenced by sideup.
27
28     def get_sideup(self):
29         return self.sideup
```

- ▶ The Coin class has three methods:
  - ▶ The `__init__` method appears in lines 11 through 12.
  - ▶ The toss method appears in lines 19 through 23.
  - ▶ The `get_sideup` method appears in lines 28 through 29.
- ▶ Except for the fact that they appear inside a class, notice these method definitions look like any other function definition in Python.
- ▶ They start with a header line, which is followed by an indented block of statements.

# self parameter

- ▶ Take a closer look at the header for each of the method definitions (lines 11, 19, and 28) and notice each method has a parameter variable named self:

- ▶ Line 11: `def __init__(self):`
- ▶ Line 19: `def toss(self):`
- ▶ Line 28: `def get_sideup(self):`

- ▶ **The self parameter is required in every method of a class.**



- ▶ Recall from our earlier discussion on object-oriented programming that a method operates on a specific object's data attributes.
- ▶ When a method executes, it must have a way of knowing which object's data attributes it is supposed to operate on.
- ▶ That's where the self parameter comes in.
- ▶ When a method is called, Python makes the self parameter reference the specific object that the method is supposed to operate on.

# Methods

- ▶ Let's look at each of the methods. The first method, which is named `__init__`, is defined in lines 11 through 12:

```
def __init__(self):  
    self.sideup = 'Heads'
```

- ▶ Most Python classes have a special method named `__init__`, which is automatically executed when an instance of the class is created in memory.



- ▶ The `__init__` method is commonly known as an **initializer** method because it initializes the object's data attributes.

- ▶ Immediately after an object is created in memory, the `__init__` method executes, and the **self parameter** is automatically assigned the object that was just created.


- ▶ The statement in line 12 assigns the string 'Heads' to the sideup data attribute belonging to the object that was just created:

```
self.sideup = 'Heads'
```



- ▶ As a result of this `__init__` method, each object we create from the Coin class will initially have a sideup attribute that is set to 'Heads'.

# Methods

- ▶ The toss method appears in lines 19 through 23.
- ▶ **This method also has the required self parameter variable.** 
- ▶ When the toss method is called, self will automatically reference the object on which the method is to operate.
- ▶ The toss method simulates the tossing of the coin.
- ▶ When the method is called, the if statement in line 20 calls the random.randint function to get a random integer in the range of 0 through 1.
- ▶ If the number is 0, then the statement in line 21 assigns 'Heads' to self.sideup. Otherwise, the statement in line 23 assigns 'Tails' to self.sideup.

# Methods

- ▶ The `get_sideup` method appears in lines 28 through 29.
- ▶ **Once again, the method has the required `self` parameter variable.**
- ▶ This method simply returns the value of `self.sideup`.
- ▶ **We call this method any time we want to know which side of the coin is facing up.**

# Driver Program

```
31 # The main function.
32 def main():
33     # Create an object from the Coin class.
34     my_coin = Coin()
35
36     # Display the side of the coin that is facing up.
37     print('This side is up:', my_coin.get_sideup())
38
39     # Toss the coin.
40     print('I am tossing the coin ...')
41     my_coin.toss()
42
43     # Display the side of the coin that is facing up.
44     print('This side is up:', my_coin.get_sideup())
45
46 # Call the main function.
47 main()
```

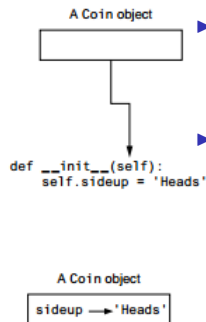
- ▶ To demonstrate the Coin class, we need to write a complete program that uses it to create an object.
- ▶ The program shows an example.
- ▶ The Coin class definition appears in lines 6 through 29. The program has a main function, which appears in lines 32 through 44.

# Driver Program

- ① An object is created in memory from the Coin class.

- ② The Coin class's `__init__` method is called, and the `self` parameter is set to the newly created object

After these steps take place, a Coin object will exist with its `sideup` attribute set to 'Heads'.



- ▶ Take a closer look at the statement in line 34:  
`my_coin = Coin()`
- ▶ The expression `Coin()` that appears on the right side of the `=` operator causes two things to happen:
  - ▶ An object is created in memory from the Coin class.
  - ▶ The Coin class's `__init__` method is executed, and the `self` parameter is automatically set to the object that was just created.
  - ▶ As a result, that object's `sideup` attribute is assigned the string 'Heads'.



# Driver Program

```
31 # The main function.
32 def main():
33     # Create an object from the Coin class.
34     my_coin = Coin()
35
36     # Display the side of the coin that is facing up.
37     print('This side is up:', my_coin.get_sideup())
38
39     # Toss the coin.
40     print('I am tossing the coin ...')
41     my_coin.toss()
42
43     # Display the side of the coin that is facing up.
44     print('This side is up:', my_coin.get_sideup())
45
46 # Call the main function.
47 main()
```

- ▶ The next statement to execute is line 37:

```
print('This side is up:',
my_coin.get_sideup())
```

- ▶ This statement prints a message indicating the side of the coin that is facing up.
- ▶ Notice the following expression appears in the statement:  
`my_coin.get_sideup()`
- ▶ This expression uses the object referenced by `my_coin` to call the `get_sideup` method.
- ▶ When the method executes, the **self parameter** will reference the `my_coin` object.
- ▶ As a result, the method returns the string 'Heads'

# Driver Program

- ▶ Notice we did not have to pass an argument to the `sideup` method, despite the fact that it has the `self` parameter variable.
- ▶ When a method is called, Python automatically passes a reference to the calling object into the method's first parameter.
- ▶ As a result, the `self` parameter will automatically reference the object on which the method is to operate.



# Driver Program

- ▶ Lines 40 and 41 are the next statements to execute:  

```
print('I am tossing the coin ...')  
my_coin.toss()
```
- ▶ The statement in line 41 uses the object referenced by `my_coin` to call the `toss` method.
- ▶ When the method executes, the `self` parameter will reference the `my_coin` object.
- ▶ The method will randomly generate a number, then use that number to change the value of the object's `sideup` attribute.

# Hiding Attributes

- ▶ Earlier in this chapter, we mentioned that an object's data attributes should be private, so that only the object's methods can directly access them.
- ▶ This protects the object's data attributes from accidental corruption.
- ▶ However, in the Coin class that was shown in the previous example, the sideup attribute is not private.
- ▶ **It can be directly accessed by statements that are not in a Coin class method.**

# Hiding Attributes

```
31 # The main function.
32 def main():
33     # Create an object from the Coin class.
34     my_coin = Coin()
35
36     # Display the side of the coin that is facing up.
37     print('This side is up:', my_coin.get_sideup())
38
39     # Toss the coin.
40     print('I am tossing the coin ...')
41     my_coin.toss()
42
43     # But now I'm going to cheat! I'm going to
44     # directly change the value of the object's
45     # sideup attribute to 'Heads'.
46     my_coin.sideup = 'Heads'
47
48     # Display the side of the coin that is facing up.
49     print('This side is up:', my_coin.get_sideup())
```

- ▶ Line 34 creates a Coin object in memory and assigns it to the my\_coin variable.
- ▶ The statement in line 37 displays the side of the coin that is facing up, then line 41 calls the object's toss method.
- ▶ Then, the statement in line 46 directly assigns the string 'Heads' to the object's sideup attribute:  
my\_coin.sideup = 'Heads'

# Hiding Attributes

**Program Output**

```
This side is up: Heads  
I am tossing the coin ...  
This side is up: Heads
```

**Program Output**

```
This side is up: Heads  
I am tossing the coin ...  
This side is up: Heads
```

**Program Output**

```
This side is up: Heads  
I am tossing the coin ...  
This side is up: Heads
```

- ▶ Regardless of the outcome of the toss method, this statement will change the my\_coin object's sideup attribute to 'Heads'.
- ▶ As you can see from the three sample runs of the program, the coin always lands heads up!

# Hiding Attributes

- ▶ If we truly want to simulate a coin that is being tossed, then we don't want code outside the class to be able to change the result of the toss method.
- ▶ To prevent this from happening, we need to make the sideup attribute private.
- ▶ In Python, you can hide an attribute by starting its name with two underscore characters.
- ▶ If we change the name of the sideup attribute to `__sideup`, then code outside the Coin class will not be able to access it.

# Hiding Attributes

```
1 import random
2
3 # The Coin class simulates a coin that can be flipped.
4 class Coin:
5
6     # The __init__ method initializes the __sideup data attribute with 'Heads'.
7     def __init__(self):
8         self.__sideup = 'Heads'
9
10    # The toss method generates a random number in the range of 0 through 1.
11    # If the number is 0, then sideup is set to 'Heads'.
12    # Otherwise, sideup is set to 'Tails'.
13    def toss(self):
14        if random.randint(0, 1) == 0:
15            self.__sideup = 'Heads'
16        else:
17            self.__sideup = 'Tails'
18
19    # The get_sideup method returns the value referenced by sideup.
20    def get_sideup(self):
21        return self.__sideup
22
23 # The main function.
24 def main():
25     # Create an object from the Coin class.
26     my_coin = Coin()
27
28     # Display the side of the coin that is facing up.
29     print('This side is up:', my_coin.get_sideup())
30
31     # Toss the coin.
32     print('I am going to toss the coin ten times:')
33     for count in range(10):
34         my_coin.toss()
35         print(my_coin.get_sideup())
36
37 # Call the main function.
38 main()
```



# Storing Classes in Modules

- ▶ The programs you have seen so far in this chapter have the Coin class definition in the same file as the programming statements that use the Coin class.
- ▶ This approach works fine with small programs that use only one or two classes. As programs use more classes, however, the need to organize those classes becomes greater.
- ▶ Programmers commonly organize their class definitions by storing them in **modules**. Then the modules can be imported into any programs that need to use the classes they contain.

# Storing Classes in Modules

```
1 import random
2
3 # The Coin class simulates a coin that can be flipped.
4 class Coin:
5
6     # The __init__ method initializes the __sideup data attribute with 'Heads'.
7     def __init__(self):
8         self.__sideup = 'Heads'
9
10    # The toss method generates a random number in the range of 0 through 1.
11    # If the number is 0, then sideup is set to 'Heads'.
12    # Otherwise, sideup is set to 'Tails'.
13    def toss(self):
14        if random.randint(0, 1) == 0:
15            self.__sideup = 'Heads'
16        else:
17            self.__sideup = 'Tails'
18
19    # The get_sideup method returns the value referenced by sideup.
20    def get_sideup(self):
21        return self.__sideup
22
23 |
```

# Storing Classes in Modules

```
1
2 # This program imports the coin module and
3 # creates an instance of the Coin class.
4
5 import coin
6
7 # The main function.
8 def main():
9     # Create an object from the Coin class.
10    my_coin = coin.Coin()
11
12    # Display the side of the coin that is facing up.
13    print('This side is up:', my_coin.get_sideup())
14
15    # Toss the coin.
16    print('I am going to toss the coin ten times:')
17    for count in range(10):
18        my_coin.toss()
19        print(my_coin.get_sideup())
20
21 # Call the main function.
22 main()
```

- ▶ Line 4 imports the coin module.
- ▶ Notice in line 8, we had to qualify the name of the Coin class by prefixing it with the name of the module, followed by a dot:

```
my_coin = coin.Coin()
```



# The BankAccount Class

```
1 # The BankAccount class simulates a bank account.
2
3 class BankAccount:
4
5     # The __init__ method accepts an argument for
6     # the account's balance. It is assigned to
7     # the __balance attribute.
8
9     def __init__(self, bal):
10         self.__balance = bal
11
12     # The deposit method makes a deposit into the
13     # account.
14
15     def deposit(self, amount):
16         self.__balance += amount
17
18     # The withdraw method withdraws an amount
19     # from the account.
20
21     def withdraw(self, amount):
22         if self.__balance >= amount:
23             self.__balance -= amount
24         else:
25             print('Error: Insufficient funds')
26
27     # The get_balance method returns the
28     # account balance.
29
30     def get_balance(self):
31         return self.__balance
```

- ▶ Notice the `__init__` method has two parameter variables: `self` and `bal`.
- ▶ The `bal` parameter will accept the account's starting balance as an argument.
- ▶ In line 10, the `bal` parameter amount is assigned to the object's `__balance` attribute.
- ▶ The `deposit` method is in lines 15 through 16. This method has two parameter variables: `self` and `amount`.
- ▶ When the method is called, the amount that is to be deposited into the account is passed into the `amount` parameter.
- ▶ The value of the parameter is then added to the `__balance` attribute in line 16.

# The BankAccount Class

```
1 # The BankAccount class simulates a bank account.
2
3 class BankAccount:
4
5     # The __init__ method accepts an argument for
6     # the account's balance. It is assigned to
7     # the __balance attribute.
8
9     def __init__(self, bal):
10         self.__balance = bal
11
12     # The deposit method makes a deposit into the
13     # account.
14
15     def deposit(self, amount):
16         self.__balance += amount
17
18     # The withdraw method withdraws an amount
19     # from the account.
20
21     def withdraw(self, amount):
22         if self.__balance >= amount:
23             self.__balance -= amount
24         else:
25             print('Error: Insufficient funds')
26
27     # The get_balance method returns the
28     # account balance.
29
30     def get_balance(self):
31         return self.__balance
```

- ▶ The withdraw method has two parameter variables: self and amount.
- ▶ When the method is called, the amount that is to be withdrawn from the account is passed into the amount parameter.
- ▶ The if statement that begins in line 22 determines whether there is enough in the account balance to make the withdrawal. If so, amount is subtracted from \_\_balance in line 23. Otherwise, line 25 displays the message 'Error: Insufficient funds'.
- ▶ The get\_balance method returns the value of the \_\_balance attribute.

# The BankAccount Driver Program

```
1 # This program demonstrates the BankAccount class.
2
3 import bankaccount
4
5 def main():
6     # Get the starting balance.
7     start_bal = float(input('Enter your starting balance: '))
8
9     # Create a BankAccount object.
10    savings = bankaccount.BankAccount(start_bal)
11
12    # Deposit the user's paycheck.
13    pay = float(input('How much were you paid this week? '))
14    print('I will deposit that into your account.')
15    savings.deposit(pay)
16
17    # Display the balance.
18    print('Your account balance is $',
19          format(savings.get_balance(), '.2f'),
20          sep='')
21
22    # Get the amount to withdraw.
23    cash = float(input('How much would you like to withdraw? '))
24    print('I will withdraw that from your account.')
25    savings.withdraw(cash)
26
27    # Display the balance.
28    print('Your account balance is $',
29          format(savings.get_balance(), '.2f'),
30          sep='')

```

- ▶ Line 7 gets the starting account balance from the user and assigns it to the `start_bal` variable.
- ▶ Line 10 creates an instance of the `BankAccount` class and assigns it to the `savings` variable.
- ▶ Take a closer look at the statement:  
`savings = bankaccount.BankAccount(start_bal)`
- ▶ Notice the `start_bal` variable is listed inside the parentheses. This causes the `start_bal` variable to be passed as an argument to the `__init__` method.
- ▶ In the `__init__` method, it will be passed into the `bal` parameter.

# The BankAccount Driver Program

```
1 # This program demonstrates the BankAccount class.
2
3 import bankaccount
4
5 def main():
6     # Get the starting balance.
7     start_bal = float(input('Enter your starting balance: '))
8
9     # Create a BankAccount object.
10    savings = bankaccount.BankAccount(start_bal)
11
12    # Deposit the user's paycheck.
13    pay = float(input('How much were you paid this week? '))
14    print('I will deposit that into your account.')
15    savings.deposit(pay)
16
17    # Display the balance.
18    print('Your account balance is $',
19          format(savings.get_balance(), '.2f'),
20          sep='')
21
22    # Get the amount to withdraw.
23    cash = float(input('How much would you like to withdraw? '))
24    print('I will withdraw that from your account.')
25    savings.withdraw(cash)
26
27    # Display the balance.
28    print('Your account balance is $',
29          format(savings.get_balance(), '.2f'),
30          sep='')

```

- ▶ Line 13 gets the amount of the user's pay and assigns it to the pay variable.
- ▶ In line 15, the savings.deposit method is called, passing the pay variable as an argument. In the deposit method, it will be passed into the amount parameter.
- ▶ The statement in lines 18 through 20 displays the account balance. It displays the value returned from the savings.get\_balance method.
- ▶ Line 23 gets the amount that the user wants to withdraw and assigns it to the cash variable.
- ▶ In line 25 the savings.withdraw method is called, passing the cash variable as an argument. In the withdraw method, it will be passed into the amount parameter.

# The `__str__` Method

- ▶ Quite often, we need to display a message that indicates an object's state.
- ▶ An object's state is simply the values of the object's attributes at any given moment.
- ▶ For example, recall the `BankAccount` class has one data attribute: `__balance`. At any given moment, a `BankAccount` object's `__balance` attribute will reference some value.
- ▶ The value of the `__balance` attribute represents the object's state at that moment.
- ▶ The following might be an example of code that displays a `BankAccount` object's state:

```
account = bankaccount.BankAccount(1500.0)
print('The balance is ', format(savings.get_balance(),
',.2f'), sep="")
```



# The `__str__` Method

- ▶ Displaying an object's state is a common task.
- ▶ It is so common that many programmers equip their classes with a method that returns a string containing the object's state.
- ▶ In Python, you give this method the special name `__str__`.
- ▶ The `__str__` method appears in lines 36 through 37. It returns a string indicating the account-balance.

```
33     # The __str__ method returns a string
34     # indicating the object's state.
35
36     def __str__(self):
37         return 'The balance is $' + format(self.__balance, ',.2f')
```

# The `__str__` Method

```
1 # This program demonstrates the BankAccount class
2 # with the __str__ method added to it.
3
4 import bankaccount2
5
6 def main():
7     # Get the starting balance.
8     start_bal = float(input('Enter your starting balance: '))
9
10    # Create a BankAccount object.
11    savings = bankaccount2.BankAccount(start_bal)
12
13    # Deposit the user's paycheck.
14    pay = float(input('How much were you paid this week? '))
15    print('I will deposit that into your account.')
16    savings.deposit(pay)
17
18    # Display the balance.
19    print(savings)
20
21    # Get the amount to withdraw.
22    cash = float(input('How much would you like to withdraw? '))
23    print('I will withdraw that from your account.')
24    savings.withdraw(cash)
25
26    # Display the balance.
27    print(savings)
28
29 # Call the main function.
30 main()
```



- ▶ You do not directly call the `__str__` method. Instead, it is automatically called when you pass an object as an argument to the print function.
- ▶ The name of the object, `savings`, is passed to the print function in lines 19 and 27.
- ▶ This causes the `BankAccount` class's `__str__` method to be called. The string that is returned from the `__str__` method is then displayed.

# The `__str__` Method

- ▶ The `__str__` method is also called automatically when an object is passed as an argument to the built-in `str` function.
- ▶ Here is an example:

```
account = bankaccount2.BankAccount(1500.0)
message = str(account)
print(message)
```
- ▶ In the second statement, the `account` object is passed as an argument to the `str` function. This causes the `BankAccount` class's `__str__` method to be called.
- ▶ The string that is returned is assigned to the `message` variable then displayed by the `print` function in the third line.