**Conditions and Branching**

Comparison operations compares some value or operand. Then based on some condition, they produce a Boolean. Let's say we assign a value of a to six. We can use the equality operator denoted with two equal signs to determine if two values are equal. In this case, if seven is equal to six. In this case, as six is not equal to seven, the result is false. If we performed an equality test for the value six, the two values would be equal. As a result, we would get a true. Consider the following equality comparison operator: If the value of the left operand, in this case, the variable i is greater than the value of the right operand, in this case five, the condition becomes true or else we get a false. Let's display some values for i on the left. Let's see the value is greater than five in green and the rest in red. If we set i equal to six, we see that six is larger than five and as a result, we get a true. We can also apply the same operations to floats. If we modify the operator as follows, if the left operand i is greater than or equal to the value of the right operand, in this case five, then the condition becomes true. In this case, we include the value of five in the number line and the color changes to green accordingly. If we set the value of i equal to five, the operand will produce a true. If we set the value of i to two, we would get a false because two is less than five. We can change the inequality if the value of the left operand, in this case, i is less than the value of the right operand, in this case, six. Then condition becomes true. Again, we can represent this with a colored number line. The areas where the inequality is true are marked in green and red where the inequality is false. If the value for i is set to two, the result is a true. As two is less than six. The inequality test uses an explanation mark preceding the equal sign. If two operands are not equal, then the condition becomes true. We can use a number line. When the condition is true, the corresponding numbers are marked in green and red for where the condition is false. If we set i equal to two, the operator is true as two is not equal to six. We compare strings as well. Comparing ACDC and Michael Jackson using the equality test, we get a false, as the strings are not the same. Using the inequality test, we get a true, as the strings are different. See the Lapps for more examples. Branching allows us to run different statements for a different input. It's helpful to think of an if statement as a locked room. If this statement is true, you can enter the room and your program can run some predefined task. If the statement is false, your program will skip the task. For example, consider the blue rectangle representing an ACDC concert. If the individual is 18 or older, they can enter the ACDC concert. If they are under the age of 18, they cannot enter the concert. Individual proceeds to the concert their age is 17, therefore, they are not granted access to the concert and they must move on. If the individual is 19, the condition is true. They can enter the concert then they can move on. This is the syntax of the if statement from our previous example. We have the if statement. We have the expression that can be true or false. The brackets are not necessary. We have a colon. Within an indent, we have the expression that is run if the condition is true. The statements after the if statement will run regardless if the condition is true or false. For the case where the age is 17, we set the value of the variable age to 17. We check the if statement, the statement is false. Therefore the program will not execute the statement to print, "you will enter". In this case, it will just print "move on". For the case where the age is 19, we set the value of the variable age to 19. We check the if statement. The statement is true. Therefore, the program will execute the statement to print "you will enter". Then it will just print "move on". The else statement will run a different block of code if the same condition is false. Let's use the ACDC concert analogy again. If the user is 17, they cannot go to the ACDC concert but they can go to the Meat Loaf concert represented by the purple square. If the individual is 19, the condition is true, they can enter the ACDC concert then they can move on as before. The syntax of the else statement is similar. We simply append the statement else. We then add the expression we would like to execute with an indent. For the case where the age is 17, we set the value of the variable age to 17. We check the if statement, the statement is false. Therefore, we progress to the else statement. We run the statement in the indent. This corresponds to the individual attending the Meat Loaf concert. The program will then continue running. For the case where the age is 19, we set the value of the variable age to 19. We check the if statement, the statement is true. Therefore, the program will execute the statement to print "you will enter". The program skips the expressions in the else statement and continues to run the rest of the expressions. The elif statement, short for else if, allows us to check additional conditions if the preceding condition is false. If the condition is true, the alternate expressions will be run. Consider the concert example, if the individual is 18, they will go to the Pink Floyd concert instead of attending the ACDC or Meat Loaf concerts. The person of 18 years of age enters the area as they are not over 19 years of age. They cannot see ACDC but as their 18 years, they attend Pink Floyd. After seeing Pink Floyd, they move on. The syntax of the elif statement is similar. We simply add the statement elif with the condition. We, then add the expression we would like to execute if the statement is true with an indent. Let's illustrate the code on the left. An 18 year old enters. They are not older than 18 years of age. Therefore, the condition is false. So the condition of the elif statement is checked. The condition is true. So then we would print "go see Pink Floyd". Then we would move on as before. If the variable age was 17, the statement "go see Meat Loaf" would print. Similarly, if the age was greater than 18, the statement "you can enter" would print. Check the Lapps for more examples. Now let's take a look at logic operators. Logic operations take Boolean values and produce different Boolean values. The first operation is the not operator. If the input is true, the result is a false. Similarly, if the input is false, the result is a true. Let A and B represent Boolean variables. The OR operator takes in the two values and produces a new Boolean value. We can use this table to represent the different values. The first column represents the possible values of A. The second column represents the possible values of B. The final column represents the result of applying the OR operation. We see the OR operator only produces a false if all the Boolean values are false. The following lines of code will print out: "This album was made in the 70s' or 90's", if the variable album year does not fall in the 80s. Let's see what happens when we set the album year to 1990. The colored number line is green when the condition is true and red when the condition is false. In this case, the condition is false. Examining the second condition, we see that 1990 is greater than 1989. So the condition is true. We can verify by examining the corresponding second number line. In the final number line, the green region indicates, where the area is true. This region corresponds to where at least one statement is true. We see that 1990 falls in the area. Therefore, we execute the statement. Let A and B represent Boolean variables. The AND operator takes in the two values and produces a new Boolean value. We can use this table to represent the different values. The first column represents the possible values of A. The second column represents the possible values of B. The final column represents the result of applying the AND operation. We see the OR operator only produces a true if all the Boolean values are true. The following lines of code will print out "This album was made in the 80's" if the variable album year is between 1980 and 1989. Let's see what happens when we set the album year to 1983. As before, we can use the colored number line to examine where the condition is true. In this case, 1983 is larger than 1980, so the condition is true. Examining the second condition, we see that 1990 is greater than 1983. So, this condition is also true. We can verify by examining the corresponding second number line. In the final number line, the green region indicates where the area is true. Similarly, this region corresponds to where both statements are true. We see that 1983 falls in the area. Therefore, we execute the statement. Branching allows us to run different statements for different inputs.

**Loops**

Before we talk about loops, let's go over the range function. The range function outputs and ordered sequence as a list I. If the input is a positive integer, the output is a sequence. The sequence contains the same number of elements as the input but starts at zero. For example, if the input is three the output is the sequence zero, one, two. If the range function has two inputs where the first input is smaller than the second input, the output is a sequence that starts at the first input. Then the sequence iterates up to but not including the second number. For the input 10 and 15 we get the following sequence. See the labs for more capabilities of the range function. Please note, if you use Python three, the range function will not generate a list explicitly like in Python two. In this section, we will cover for loops. We will focus on lists, but many of the procedures can be used on tuples. Loops perform a task over and over. Consider the group of colored squares. Let's say we would like to replace each colored square with a white square. Let's give each square a number to make things a little easier and refer to all the group of squares as squares. If we wanted to tell someone to replace squares zero with a white square, we would say equals replace square zero with a white square or we can say four squares zero in squares square zero equals white square. Similarly, for the next square we can say for square one in squares, square one equals white square. For the next square we can say for square two in squares, square two equals white square. We repeat the process for each square. The only thing that changes is the index of the square we are referring to. If we're going to perform a similar task in Python we cannot use actual squares. So let's use a list to represent the boxes. Each element in the list is a string representing the color. We want to change the name of the color in each element to white. Each element in the list has the following index. This is a syntax to perform a loop in Python. Notice the indent, the range function generates a list. The code will simply repeat everything in the indent five times. If you were to change the value to six it would do it 6 times. However, the value of I is incremented by one each time. In this segment we change the I element of the list to the string white. The value of I is set to zero. Each iteration of the loop starts at the beginning of the indent. We then run everything in the indent. The first element in the list is set to white. We then go to the start of the indent, we progress down each line. When we reach the line to change the value of the list, we set the value of index one to white. The value of I increases by one. We repeat the process for index two. The process continues for the next index, until we've reached the final element. We can also iterate through a list or tuple directly in python, we do not even need to use indices. Here is the list squares. Each iteration of the list we pass one element of the list squares to the variable square. Lets display the value of the variable square on this section. For the first iteration, the value of square is red, we then start the second iteration. For the second iteration, the value of square is yellow. We then start the third iteration. For the final iteration, the value of square is green, a useful function for iterating data is enumerate. It can be used to obtain the index and the element in the list. Let's use the box analogy with the numbers representing the index of each square. This is the syntax to iterate through a list and provide the index of each element. We use the list squares and use the names of the colors to represent the colored squares. The argument of the function enumerate is the list. In this case squares the variable I is the index and the variable square is the corresponding element in the list. Let's use the left part of the screen to display the different values of the variable square and I for the various iterations of the loop. For the first iteration, the value of the variable is red corresponding to the zeroth index, and the value for I is zero for the second iteration. The value of the variable square is yellow, and the value of I corresponds to its index i.e. 1. We repeat the process for the last index. While loops are similar to for loops but instead of executing a statement a set number of times a while loop will only run if a condition is met. Let's say we would like to copy all the orange squares from the list squares to the list New squares. But we would like to stop if we encounter a non-orange square. We don't know the value of the squares beforehand. We would simply continue the process while the square is orange or see if the square equals orange. If not, we would stop. For the first example, we would check if the square was orange. It satisfies the conditions so we would copy the square. We repeat the process for the second square. The condition is met. So we copy the square. In the next iteration, we encounter a purple square. The condition is not met. So we stop the process. This is essentially what a while loop does. Let's use the figure on the left to represent the code. We will use a list with the names of the color to represent the different squares. We create an empty list of new squares. In reality the list is of indeterminate size. We start the index at zero the while statement will repeatedly execute the statements within the indent until the condition inside the bracket is false. We append the value of the first element of the list squares to the list new squares. We increase the value of I by one. We append the value of the second element of the list squares to the list new squares. We increment the value of I. Now the value in the array squares is purple; therefore, the condition for the while statement is false and we exit the loop.

**Functions**

Functions take some input then produce some output or change. The function is just a piece of code you can reuse. You can implement your own function, but in many cases, you use other people’s functions. In this case, you just have to know how the function works and in some cases how to import the functions. Let the orange and yellow squares represent similar blocks of code. We can run the code using some input and get an output. If we define a function to do the task we just have to call the function. Let the small squares represent the lines of code used to call the function. We can replace these long lines of code by just calling the function a few times. Now we can just call the function; our code is much shorter. The code performs the same task. You can think of the process like this: when we call the function f1, we pass an input to the function. These values are passed to all those lines of code you wrote. This returns a value; you can use the value. For example, you can input this value to a new function f2. When we call this new function f2, the value is passed to another set of lines of code. The function returns a value. The process is repeated passing the values to the function you call. You can save these functions and reuse them, or use other people’s functions. Python has many built-in functions; you don't have to know how those functions work internally, but simply what task those functions perform. The function len takes in an input of type sequence, such as a string or list, or type collection, such as a dictionary or set, and returns the length of that sequence or collection. Consider the following list. The len function takes this list as an argument, and we assign the result to the variable L. The function determines there are 8 items in the list, then returns the length of the list, in this case, 8. The function sum takes in an iterable like a tuple or list and returns the total of all the elements. Consider the following list. We pass the list into the sum function and assign the result to the variable S. The function determines the total of all the elements, then returns it, in this case, the value is 70. There are two ways to sort a list. The first is using the function sorted. We can also use the list method sort. Methods are similar to functions. Let's use this as an example to illustrate the difference. The function sorted Returns a new sorted list or tuple. Consider the list album ratings. We can apply the function sorted to the list album ratings and get a new list sorted album rating. The result is a new sorted list. If we look at the list album ratings, nothing has changed. Generally, functions take an input, in this case, a list. They produce a new output, in this instance, a sorted list. If we use the method sort, the list album ratings will change and no new list will be created. Let's use this diagram to help illustrate the process. In this case, the rectangle represents the list album ratings. When we apply the method sort to the list, the list album rating changes. Unlike the previous case, we see that the list album rating has changed. In this case, no new list is created. Now that we have gone over how to use functions in Python, let’s see how to build our own functions. We will now get you started on building your own functions in python. This is an example of a function in python that returns its input value + 1. To define a function, we start with the keyword def. The name of the function should be descriptive of what it does. We have the function formal parameter "A" in parentheses. Followed by a colon. We have a code block with an indent, for this case, we add 1 to "A" and assign it to B. We return or output the value for b. After we define the function, we can call it. The function will add 1 to 5 and return a 6. We can call the function again; this time assign it to the variable "c" The value for 'c' is 11. Let's explore this further. Let's go over an example when you call a function. It should be noted that this is a simplified model of Python, and Python does not work like this under the hood. We call the function giving it an input, 5. It helps to think of the value of 5 as being passed to the function. Now the sequences of commands are run, the value of "A" is 5. "B" would be assigned a value of 6. We then return the value of b, in this case, as b was assigned a value of 6, the function returns a 6. If we call the function again, the process starts from scratch; we pass in an 8. The subsequent operations are performed. Everything that happened in the last call will happen again with a different value of "A" The function returns a value, in this case, 9. Again, this is just a helpful analogy. Let’s try and make this function more complex. It's customary to document the function on the first few lines; this tells anyone who uses the function what it does. This documentation is surrounded in triple quotes. You can use the help command on the function to display the documentation as follows. This will printout the function name and the documentation. We will not include the documentation in the rest of the examples. A function can have multiple parameters. The function mult multiplies two numbers; in other words, it finds their product. If we pass the integers 2 and 3, the result is a new integer. If we pass the integer 10 and the float 3.14, the result is a float 31.4. If we pass in the integer two and the string “Michael Jackson,” the string Michael Jackson is repeated two times. This is because the multiplication symbol can also mean repeat a sequence. If you accidentally multiply an integer and a String instead of two integers, you won’t get an error. Instead, you will get a String, and your program will progress, potentially failing later because you have a String where you expected an integer. This property will make coding simpler, but you must test your code more thoroughly. In many cases a function does not have a return statement. In these cases, Python will return the special “None” object. Practically speaking, if your function has no return statement, you can treat it as if the function returns nothing at all. The function MJ simply prints the name 'Michael Jackson’. We call the function. The function prints “Michael Jackson.” Let's define the function “No work” that performs no task. Python doesn’t allow a function to have an empty body, so we can use the keyword pass, which doesn’t do anything, but satisfies the requirement of a non-empty body. If we call the function and print it out, the function returns a None. In the background, if the return statement is not called, Python will automatically return a None. It is helpful to view the function No Work with the following return statement. Usually, functions perform more than one task. This function prints a statement then returns a value. Let's use this table to represent the different values as the function is called. We call the function with an input of 2. We find the value of b. The function prints the statement with the values of a and b. Finally, the function returns the value of b, in this case, 3. We can use loops in functions. This function prints out the values and indexes of a loop or tuple. We call the function with the list album ratings as an input. Let's display the list on the right with its corresponding index. Stuff is used as an input to the function enumerate. This operation will pass the index to i and the value in the list to “s”. The function would begin to iterate through the loop. The function will print the first index and the first value in the list. We continue iterating through the loop. The values of i and s are updated. The print statement is reached. Similarly, the next values of the list and index are printed. The process is repeated. The values of i and s are updated. We continue iterating until the final values in the list are printed out. Variadic parameters allow us to input a variable number of elements. Consider the following function; the function has an asterisk on the parameter names. When we call the function, three parameters are packed into the tuple names. We then iterate through the loop; the values are printed out accordingly. If we call the same function with only two parameters as inputs, the variable names only contain two elements. The result is only two values are printed out. The scope of a variable is the part of the program where that variable is accessible. Variables that are defined outside of any function are said to be within the global scope, meaning they can be accessed anywhere after they are defined. Here we have a function that adds the string DC to the parameter x. When we reach the part where the value of x is set to AC, this is within the global scope, meaning x is accessible anywhere after it is defined. A variable defined in the global scope is called a global variable. When we call the function, we enter a new scope or the scope of AddDC. We pass as an argument to the AddDC function, in this case, AC. Within the scope of the function, the value of x is set to ACDC. The function returns the value and is assigned to z. Within the global scope, the value z is set to ACDC After the value is returned, the scope of the function is deleted. Local variables only exist within the scope of a function. Consider the function thriller; the local variable Date is set to 1982. When we call the function, we create a new scope. Within that scope of the function, the value of the date is set to 1982. The value of date does not exist within the global scope. Variables inside the global scope can have the same name as variables in the local scope with no conflict. Consider the function thriller; the local variable Date is set to 1982. The global variable date is set to 2017. When we call the function, we create a new scope. Within that scope, the value of the date is set to 1982. If we call the function, it returns the value of Date in the local scope, in this case, 1982. (click6) When we print in the global scope, we use the global variable value. The global value of the variable is 2017. Therefore, the value is set to 2017. If a variable is not defined within a function, Python will check the global scope. Consider the function "AC-DC“. The function has the variable rating, with no value assigned. If we define the variable rating in the global scope, then call the function, Python will see there is no value for the variable Rating. As a result, python will leave the scope and check if the variable Ratings exists in the global scope. It will use this value of Ratings in the global scope within the scope of "AC-DC“. In the function, will print out a 9. The value of z in the global scope will be 10, as we added one. The value of rating will be unchanged within the global scope. Consider the function Pink Floyd. If we define the variable Claimed Sales with the keyword global, the variable will be a global variable. We call the function Pink Floyd. The variable claimed sales is set to the string “45 million” in the global scope. When we print the variable, we get a value of “45 million.”

**Exception Handling**

After watching this video, you will be able to: Explain Exception Handling, Demonstrate the use of exception handling, and Understand the basics of exception handling. Have you ever mistakenly entered a number when you were supposed to enter text in an input field? Most of us have either in error or when testing out a program, but do you know why it gave an error message instead of completing and terminating the program? In order for the error message to appear an event was triggered in the background. This event was activated because the program tried to perform a computation on the name entry and realized the entry contained numbers and not letters. By encasing this code in an exception handler the program knew how to deal with this type of error and was able to output the error message to continue along with the program. This is one of many errors that can happen when asking for user input, so let’s see how exception handling works. Let’s first explore the try…except statement. This type of statement will first attempt to execute the code in the “try” block, but if an error occurs it will kick out and begin searching for the exception that matches the error. Once it finds the correct exception to handle the error it will then execute that line of code. For example, let’s say you are writing a program that will open and write a file. After starting the program an error occurred as the data was not able to be read. Because of this error the program skipped over the code lines under the “try” statement and went directly to the exception line. Since this error fell within the IOError guidelines it printed “Unable to open or read the data in the file.” to our console. When writing simple programs we can sometimes get away with only one except statement, but what happens if another error occurs that is not caught by the IOError? If that happened we would need to add another except statement. For this except statement you will notice that the type of error to catch is not specified. While this may seem a logical step so the program will catch all errors and not terminate this is not a best practice. For example, let’s say our small program was just one section of a much larger program that was over a thousand lines of code. Our task was to debug the program as it kept throwing an error causing a disruption for our users. When investigating the program you found this error kept appearing. Because this error had no details you ended up spending hours trying to pinpoint and fix the error. So far in our program we have defined that an error message should print out if an error occurs, but we don’t receive any messages that the program executed properly. This is where we can now add an else statement to give us that notification. By adding this else statement it will provide us a notification to the console that “The file was written successfully”. Now that we have defined what will happen if our program executes properly, or if an error occurs there is one last statement to add. For this example since we are opening a file the last thing we need to do is close the file. By adding a finally statement it will tell the program to close the file no matter the end result and print “File is now closed” to our console. In this video, you learned: How to write a try…except statement, Why is it important to always define errors when creating exceptions, and How to add an else and finally statement.

**Objects and Classes**

Python has many different kinds of data types: integers, floats, strings, lists, dictionaries, booleans. In Python, each is an object. Every object has the following: a type, internal representation, a set of functions called methods to interact with the data. An object is an instance of a particular type. For example, we have two types, type one and type two. We can have several objects of type one as shown in yellow. Each object is an instance of type one. We also have several objects of type two shown in green. Each object is an instance of type two. Let's do several less abstract examples. Every time we create an integer, we are creating an instance of type integer, or we are creating an integer object. In this case, we are creating five instances of type integer or five integer objects. Similarly, every time we create a list, we are creating an instance of type list, or we are creating a list object. In this case, we are creating five instances of type list or five list objects. We could find out the type of an object by using the type command. In this case, we have an object of type list, we have an object of type integer, we have an object of type string. Finally, we have an object of type dictionary. A class or type's methods are functions that every instance of that class or type provides. It's how you interact with the object. We have been using methods all this time, for example, on lists. Sorting is an example of a method that interacts with the data in the object. Consider the list ratings, the data is a series of numbers contained within the list. The method sort will change the data within the object. We call the method by adding a period at the end of the object's name, and the method's name we would like to call with parentheses. We have the rating's list represented in orange. The data contained in the list is a sequence of numbers. We call the sort method, this changes the data contained in the object. You can say it changes the state of the object. We can call the reverse method on the list, changing the list again. We call the method, reversing the order of the sequence within the object. In many cases, you don't have to know the inner workings of the class and its methods, you just have to know how to use them. Next, we will cover how to construct your own classes. You can create your own type or class in Python. In this section, you'll create a class. The class has data attributes. The class has methods. We then create instances or instances of that class or objects. The class data attributes define the class. Let's create two classes. The first class will be a circle, the second will be a rectangle. Let's think about what constitutes a circle. Examining this image, all we need is a radius to define a circle, and let's add color to make it easier to distinguish between different instances of the class later. Therefore, our class data attributes are radius and color. Similarly, examining the image in order to define a rectangle, we need the height and width. We will also add color to distinguish between instances later. Therefore, the data attributes are color, height, and width. To create the class circle, you will need to include the class definition. This tells Python you're creating your own class, the name of the class. For this course in parentheses, you will always place the term object, this is the parent of the class. For the class rectangle, we changed the name of the class, but the rest is kept the same. Classes are outlines we have to set the attributes to create objects. We can create an object that is an instance of type circle. The color data attribute is red, and the data attribute radius is four. We could also create a second object that is an instance of type circle. In this case, the color data attribute is green, and the data attribute radius is two. We can also create an object that is an instance of type rectangle. The color data attribute is blue, and the data attribute of height and width is two. The second object is also an instance of type rectangle. In this case, the color data attribute is yellow, and the height is one, and the width is three. We now have different objects of class circle or type circle. We also have different objects of class rectangle or type rectangle. Let us continue building the circle class in Python. We define our class. We then initialize each instance of the class with data attributes, radius, and color using the class constructor. The function init is a constructor. It's a special function that tells Python you are making a new class. There are other special functions in Python to make more complex classes. The radius and color parameters are used to initialize the radius and color data attributes of the class instance. The self parameter refers to the newly created instance of the class. The parameters, radius, and color can be used in the constructors body to access the values passed to the class constructor when the class is constructed. We could set the value of the radius and color data attributes to the values passed to the constructor method. Similarly, we can define the class rectangle in Python. The name of the class is different. This time, the class data attributes are color, height, and width. After we've created the class, in order to create an object of class circle, we introduce a variable. This will be the name of the object. We create the object by using the object constructor. The object constructor consists of the name of the class as well as the parameters. These are the data attributes. When we create a circle object, we call the code like a function. The arguments passed to the circle constructor are used to initialize the data attributes of the newly created circle instance. It is helpful to think of self as a box that contains all the data attributes of the object. Typing the object's name followed by a dot and the data attribute name gives us the data attribute value, for example, radius. In this case, the radius is 10. We can do the same for color. We can see the relationship between the self parameter and the object. In Python, we can also set or change the data attribute directly. Typing the object's name followed by a dot and the data attribute name, and set it equal to the corresponding value. We can verify that the color data attribute has changed. Usually, in order to change the data in an object, we define methods in the class. Let's discuss methods. We have seen how data attributes consist of the data defining the objects. Methods are functions that interact and change the data attributes, changing or using the data attributes of the object. Let's say we would like to change the size of a circle. This involves changing the radius attribute. We add a method, add radius to the class circle. The method has a function that requires the self as well as other parameters. In this case, we are going to add a value to the radius, We denote that value as r. We are going to add r to the data attribute radius. Let's see how this part of the code works when we create an object and call the add\_radius method. As before, we create an object with the object constructor. We pass two arguments to the constructor. The radius is set to two and the color is set to red. In the constructor's body, the data attributes are set. We can use the box analogy to see the current state of the object. We call the method by adding a dot followed by the method, name, and parentheses. In this case, the argument of the function is the amount we would like to add. We do not need to worry about the self parameter when calling the method. Just like with the constructor, Python will take care of that for us. In many cases, there may not be any parameters other than self specified in the method's definition. So we don't pass any arguments when calling the function. Internally, the method is called with a value of eight, and the proper self object. The method assigns a new value to self radius. This changes the object, in particular, the radius data attribute. When we call the add\_radius method, this changes the object by changing the value of the radius data attribute. We can add default values to the parameters of a class as constructor. In the labs, we also create the method called drawCircle. See the lab for the implementation of drawCircle. In the labs, we can create a new object of type circle using the constructor. The color will be red and the radius will be three. We can access the data attribute radius. We can access the attribute color. Finally, we can use the method drawCircle to draw the circle. Similarly, we can create a new object of type circle. We can access the data attribute of radius. We can access the data attribute color. We can use the method drawCircle to draw the circle. In summary, we have created an object of class circle called RedCircle with a radius attribute of three, and a color attribute of red. We also created an object of class circle called BlueCircle, with a radius attribute of 10 and a color attribute of blue. In the lab, we have a similar class for rectangle. We can create a new object of type rectangle using the constructor. We can access a data attribute of height. We can also access the data attribute of width. We could do the same for the data attribute of color. We can use the method drawRectangle to draw the rectangle. So we have a class, an object that is a realization or instantiation of that class. For example, we can create two objects of class Circle, or two objects of class Rectangle. The dir function is useful for obtaining the list of data attributes and methods associated with a class. The object you're interested in is passed as an argument. The return value is a list of the objects data attributes. The attribute surrounded by underscores are for internal use, and you shouldn't have to worry about them. The regular looking attributes are the ones you should concern yourself with. These are the objects, methods, and data attributes.