|  |
| --- |
| **What Are "For" Loops?**  Sometimes you have to do the same things over and over again. But there's an easier way to do this than typing out the same commands many times!  **Loops**  In Python, you can use loops to run (or execute) the same code a number of times. One such loop is a "for" loop.  **"For" Loops**  "For" loops execute the same code a certain number of times. They are useful when you need to count (e.g. count from 0 to 9).  In Python, the "for" loop has the following syntax:    #"number" is the number of times you want the code to repeat.  for i in range(number):  #commands  While the loop is iterating, it uses a counter variable, "i" in this case, to keep track of the number of times it has looped.  **Example 1**  Let's use a "for" loop to count from 0 to 9. Press **Play** to see the output.    # This for loop prints from 0 to 9  for i in range(10):  # print the value of i  print(i)  You can further customize your "for" loop by working with the range() specifier. The range specifier can receive up to 3 parameters, like this:  **range(start,end,increment)**  By default, **increment** is set to 1 and **start** is set to 0. But you can specify all of them yourself to construct the range you want. For example, you could count down from 9 to 0.  **Example 2**  Let's count down from 9 to 0. Click **Play** to see the output.    # start is 9, end is -1, and increment is -1  for i in range(9,-1,-1):  # print the value of i  print(i) |
| **What Is Indentation?**  Unlike most other programming languages, you must indent properly in Python for your code to work.  Indentation is not optional in Python, and you must follow the rules at all time.  Programmers also use indentation to show the structure of their code and make the code more readable. Indentation is typically used to separate the code inside loops or functions from the code that lies outside them. Use 4 white spaces to indent your code. Here's an example:    #loop to print numbers 0 to 9  for i in range(10):  # print number  print(i)    # end of loop  print("after loop")  Notice how a semicolon(:) ends the line of code right before the indentation starts and the code inside the loop is indented by 4 spaces. After the contents of the loop, you must stop indenting and start coding from the beginning of the line again. This also makes the code much easier to read and debug.  This is especially true for nested loops or conditionals, which you'll learn about soon. See the example below.    # loop to 5th rows  for j in range(5):  # loop to print 5th columns  for i in range(5):  print(i)    print(j)    The code inside the inner loop is indented more than the code in the outer loop. The more you nest, the more you need to indent.  **Do It Yourself**  Try indenting the following nested "for" loop to make the code work and to make it easier to read. Make sure you fix any other syntax issues as well!    ​  for i in range(9): for j in range(9)  print(i)  print(j) |
| **What Are Arithmetic Operators?**  In most computer program, you will need to perform **arithmetic operations** on **numbers**. Python has many arithmetic operators to help with computation.  Arithmetic operators can take both variables (such as x) and literal numbers (such as 3) and compute the result.  **Arithmetic Operators**   |  |  |  | | --- | --- | --- | | **Name** | **Operator** | **Usage** | | Addition | + | x = 8 + 4 | | Subtraction | - | y = x - 4 | | Multiplication | \* | x = 4 \* 32 | | Division | / | x = y / 4 | | Modulus | % | Returns the remainder after dividing the two numbers. if x = 8, then x % 4 is 0. | | Exponent | \*\* | x\*\*y Returns the value that is x to the y power.  Example:     If x = 9\*\*2, then **x** is equal to 81 | | Floor Division | // | x//y Returns the whole number part of the quotient of x divided by y. Example:     If x = 7//3, then **x** is equal to 2 |   **Example 1**  **Modulus Operator %**  The **modulus operator**, called **mod** for short, returns the remainder of the first number divided by the second number.  This is very useful for a lot of things, like checking if a number is odd. Press **Play** to see the output.    print("Example 4")  print("0 % 2 = "+str(0%2))  ​  print("1 % 2 = "+str(1%2))  ​  print("2 % 2 = "+str(2%2))  ​  print("3 % 2 = "+str(3%2))  ​  print("4 % 2 = "+str(4%2))  ​  print("5 % 2 = "+str(5%2))  ​ |
| **What are Nested Loops?**  In Python, you can add loops within loops. This is called nesting.  Here's an example of a nested loop:    # loop 1  for i in range(3):  # loop 2  for j in range(5):  **How do nested loops work?**  Let's take a look to see how nested loops work.  *loop1 (runs 3 times)*          loop2*(runs 5 times)*   * The loop on the outside, loop1, is called the **outer loop**. * The loop inside the outer loop, loop2, is called the **inner loop**. * Each time loop1 is executed, loop2 is executed 5 times:   + when loop1 runs for the first time, loop2 runs 5 times.   + when loop1 runs for the second time, loop2 runs 5 more times.   + when loop1 runs for the third time, loop2 runs another 5 times.   + loop1 runs a total of 3 times, while loop2 runs a total of 15 times.   **Nested "For" Loops**  In Python, you can nest "for" loops by adding a "for" loop inside another "for" loop as shown in the example above.  While nesting "for" loops, you should use a different variable for each "for" loop. Take a look at this example.  Here, we want to print numbers from 0 to 8 nine times. The outer loop needs to run 9 times. Each time the inner loop also runs 9 times for a total of 81 numbers.  Press the **Play** button to see the output.    # outer for loop  for i in range(9):    # inner for loop  for j in range(9):  print(i, j)  **What are Nested Loops?**  In Python, you can add loops within loops. This is called nesting.  Here's an example of a nested loop:    # loop 1  for i in range(3):  # loop 2  for j in range(5):  **How do nested loops work?**  Let's take a look to see how nested loops work.  *loop1 (runs 3 times)*          loop2*(runs 5 times)*   * The loop on the outside, loop1, is called the **outer loop**. * The loop inside the outer loop, loop2, is called the **inner loop**. * Each time loop1 is executed, loop2 is executed 5 times:   + when loop1 runs for the first time, loop2 runs 5 times.   + when loop1 runs for the second time, loop2 runs 5 more times.   + when loop1 runs for the third time, loop2 runs another 5 times.   + loop1 runs a total of 3 times, while loop2 runs a total of 15 times.   **Nested "For" Loops**  In Python, you can nest "for" loops by adding a "for" loop inside another "for" loop as shown in the example above.  While nesting "for" loops, you should use a different variable for each "for" loop. Take a look at this example.  Here, we want to print numbers from 0 to 8 nine times. The outer loop needs to run 9 times. Each time the inner loop also runs 9 times for a total of 81 numbers.  Press the **Play** button to see the output.    # outer for loop  for i in range(9):    # inner for loop  for j in range(9):  print(i, j) |
| **Review**  Let's review what you have learned so far in this lesson. Use this as a guide for the quiz coming up next.  **Loops**  In programming, loops are used to repeat the same code over and over again.  **"For" Loops**  The syntax for "for" loops in Python is as follows:  for counter in range(number):     commands()  or  for counter in range(start, end, increment):  commands()  The **counter** is a variable that holds the number of times the loop has executed. It starts from 0 by default.  You can construct the range by passing in a single number that tells it how many times to loop or you can pass in a **start**, an **end**, and an **increment**. If you pass in all three values, the counter variable will count from start to endwith the given increment.  **Nested "For" Loops**  In Python, you can nest "for" loops by adding a "for" loop inside another "for" loop as shown below.  # loop 1  for j in range(3):     # loop 2     for i in range(5):  *loop1 (runs 3 times)*          loop2*(runs 5 times)*   * The outside loop, loop1, is called the **outer loop**. * The inside loop, loop2, is called the **inner loop**. * Each loop needs a different counter variable (loop1 uses j and loop 2 uses i) * Each time loop1 is executed, loop2 is executed 5 times:   + loop1 runs first time, loop2 runs 5 times.   + loop1 runs second time, loop2 runs 5 more times.   + loop1 runs third time, loop2 runs another 5 times.   + loop1 runs a total of 3 times, loop2 runs a total of 15 times.   **Arithmetic Operators**  Arithmetic operators can take variables (such as x) or literal numbers (such as 3) and compute the result.  Here is a list of Python arithmetic operators:   |  |  |  | | --- | --- | --- | | **Name** | **Operator** | **Usage** | | Addition | + | x = 8 + 4 | | Subtraction | - | y = x - 4 | | Multiplication | \* | x = 4 \* 32 | | Division | / | x = y / 4 | | Modulus | % | Returns the remainder after dividing the two numbers. if x = 8, then x % 4 is 0. | | Exponent | \*\* | x\*\*y Returns the value that is x to the y power.  Example:     If x = 9\*\*2, then **x** is equal to 81 | | Floor Division | // | x//y Returns the whole number part of the quotient of x divided by y. Example:     If x = 7//3, then **x** is equal to 2 |   **Indentation**  Proper indentation is required for code to work properly in Python. Good indentation also shows the structure of the code.  Indentation separates code that lies inside a loop, function, or conditional and the code that lies outside.  You can use a tab or 4 white spaces to indent, but you cannot mix them together. For this reason, you should use 4 white spaces throughout this course.  Example:  # loop to print numbers 1 to 10  for i in range(10):  # print number  print(i)  # end of loop  The code inside the loop is indented more than the code outside the loop. The more you nest, the more you need to indent to show the code structure. |
|  |
|  |