**Welcome**

## Syntax

We're going to start our journey into the language of Python 3 with a lesson on the fundamental syntax.

In this lesson, we will be covering the building blocks of the programming language and teach you how to define variables and print output. These building blocks will open the world of programming. We'll also cover the writing of comments, a useful tool for explaining code in plain English.

Python is a programming language. Like other languages, it gives us a way to communicate ideas. In the case of a programming language, these ideas are "commands" that people use to communicate with a computer!

We convey our commands to the computer by writing them in a text file using a programming language. These files are called *programs*. Running a program means telling a computer to read the text file, translate it to the set of operations that it understands, and perform those actions.

Instructions

Change Codecademy to your name in the script to the right. Run the code to see what it does! As soon as you're ready, move on to the next exercise to begin learning to write your own Python programs!

my\_name = "Jim"

print ("Hello and welcome " + my\_name + "!")

**Comments**

Ironically, the first thing we're going to do is show how to tell a computer to ignore a part of a program. Text written in a program but not run by the computer is called a *comment*. Python interprets anything after a # as a comment.

Comments can:

* Provide context for why something is written the way it is:
* # This variable will be used to count the number of times anyone tweets the word persnickety

persnickety\_count = 0

* Help other people reading the code understand it faster:
* # This code will calculate the likelihood that it will rain tomorrow

complicated\_rain\_calculation\_for\_tomorrow()

* Ignore a line of code and see how a program will run without it:
* # useful\_value = old\_sloppy\_code()

useful\_value = new\_clean\_code()

Instructions

1. Documentation is an important step in programming. Write a comment describing the first program you want to write!
2. Start your comment with # and write the rest of your comment afterwards.

# This is my first program and I want to write a program on test automation

# Print

Now what we're going to do is teach our computer to communicate. The gift of speech is valuable: a computer can answer many questions we have about "how" or "why" or "what" it is doing. In Python, the print() function is used to tell a computer to talk. The message to be printed should be surrounded by quotes:

# from Mary Shelley's Frankenstein

print("There is something at work in my soul, which I do not understand.")

In the above example, we direct our program to print() an excerpt from a notable book. The printed words that appear as a result of the print() function are referred to as output. The output of this example program would be:

There is something at work in my soul, which I do not understand.

Instructions

1. Print the distinguished greeting "Hello world!"

print ("Hello World!")

# Strings

Computer programmers refer to blocks of text as strings. In our last exercise, we created the string "Hello world!". In Python a string is either surrounded by double quotes ("Hello world") or single quotes ('Hello world'). It doesn't matter which kind you use, just be consistent.

Instructions

1. Print your name using the print() command.

print ("Jim")

# Variables

Programming languages offer a method of storing data for reuse. If there is a greeting we want to present, a date we need to reuse, or a user ID we need to remember we can create a variable which can store a value. In Python, we assign variables by using the equals sign (=).

message\_string = "Hello there"

# Prints "Hello there"

print(message\_string)

In the above example, we store the message "Hello there" in a variable called message\_string. Variables can't have spaces or symbols in their names other than an underscore (\_). They can't begin with numbers, but they can have numbers after the first letter (e.g., cool\_variable\_5 is OK).

It's no coincidence we call these creatures "variables". If the context of a program changes, we can update a variable but perform the same logical process on it.

# Greeting

message\_string = "Hello there"

print(message\_string)

# Farewell

message\_string = "Hasta la vista"

print(message\_string)

Above, we create the variable message\_string, assign a welcome message, and print the greeting. After we greet the user, we want to wish them goodbye. We then update message\_string to a departure message and print that out.

Instructions

1. Update the variable meal to reflect each meal of the day before we print it.

# We've defined the variable "meal" here to the name of the food we ate for breakfast!

meal = "An english muffin"

# Printing out breakfast

print("Breakfast:")

print(meal)

# Now update meal to be lunch!

meal = "Hamburger and Fries"

# Printing out lunch

print("Lunch:")

print(meal)

# Now update "meal" to be dinner

meal = "Salmon and Veggies"

# Printing out dinner

print("Dinner:")

print(meal)

**Errors**

Humans are prone to making mistakes. Humans are also typically in charge of creating computer programs. To compensate, programming languages attempt to understand and explain mistakes made in their programs.

Python refers to these mistakes as *errors* and will point to the location where an error occurred with a ^ character. When programs throw errors that we didn't expect to encounter we call those errors *bugs*. Programmers call the process of updating the program so that it no longer produces unexpected errors *debugging*.

Two common errors that we encounter while writing Python are SyntaxError and NameError.

* SyntaxError means there is something wrong with the way your program is written — punctuation that does not belong, a command where it is not expected, or a missing parenthesis can all trigger a SyntaxError.
* A NameError occurs when the Python interpreter sees a word it does not recognize. Code that contains something that looks like a variable but was never defined will throw a NameError.

Instructions

**1.**

You might encounter a SyntaxError if you open a string with double quotes and end it with a single quote. Update the string so that it starts and ends with the same punctuation.

You might encounter a NameError if you try to print a string but fail to put any quotes around it. Python expects the words of your string to be defined elsewhere but can't find where they're defined. Add quotes to either side of the string to squash this bug.

Update the malformed strings in the workspace to all be strings.

print("This message has mismatched quote marks!")

print("This message has no quote marks!")

# Numbers

Computers can understand much more than just strings of text. Python has a few numeric data types. It has multiple ways of storing numbers. Which one you use depends on your intended purpose for the number you are saving.

An integer, or int, is a whole number. It has no decimal point and contains all counting numbers (1, 2, 3, ...) as well as their negative counterparts and the number 0. If you were counting the number of people in a room, the number of jellybeans in a jar, or the number of keys on a keyboard you would likely use an integer.

A floating-point number, or a float, is a decimal number. It can be used to represent fractional quantities as well as precise measurements. If you were measuring the length of your bedroom wall, calculating the average test score of a seventh-grade class, or storing a baseball player's batting average for the 1998 season you would likely use a float.

Numbers can be assigned to variables or used literally in a program:

an\_int = 2

a\_float = 2.1

print(an\_int + 3)

# prints 5

Above we defined an integer and a float as the variables an\_int and a\_float. We printed out the sum of the variable an\_int with the number 3. We call the number 3 here a literal, meaning it's actually the number 3 and not a variable with the number 3 assigned to it.

Floating-point numbers can behave in some unexpected ways due to how computers store them. For more information on floating-point numbers and Python, review [Python's documentation on floating-point limitations](https://docs.python.org/3/tutorial/floatingpoint.html).

**1.**

A recent movie-going experience has you excited to pub

lish a review. You rush out of the cinema and hastily begin programming to create your movie-review website: The Big Screen's Greatest Scenes Decided By A Machine.

Create the following variables and assign integer numbers to them: release\_year and runtime.

Remember, integers are any whole numbers, positive or negative!

1. Now, create the variable rating\_out\_of\_10 and assign it a float number between one and ten.

release\_year = 2018

runtime = 120

rating\_out\_of\_10 = 8.8

# Calculations

Computers absolutely excel at performing calculations. The "compute" in their name comes from their historical association with providing answers to mathematical questions. Python performs addition, subtraction, multiplication, and division with +, -, \*, and /.

# Prints "500"

print(573 - 74 + 1)

# Prints "50"

print(25 \* 2)

# Prints "2.0"

print(10 / 5)

Notice that when we perform division, the result has a decimal place. This is because Python converts all ints to floats before performing division. In older versions of Python (2.7 and earlier) this conversion did not happen, and integer division would always round down to the nearest integer.

Division can throw its own special error: ZeroDivisionError. Python will raise this error when attempting to divide by 0.

Mathematical operations in Python follow the standard mathematical [order of operations](https://en.wikipedia.org/wiki/Order_of_operations).

Instructions

1. Print out the result of this equation: 25 \* 68 + 13 / 28

print (25\*68+13/28)

# Changing Numbers

Variables that are assigned numeric values can be treated the same as the numbers themselves. Two variables can be added together, divided by 2, and multiplied by a third variable without Python distinguishing between the variables and literals (like the number 2 in this example). Performing arithmetic on variables does not change the variable — you can only update a variable using the = sign.

coffee\_price = 1.50

number\_of\_coffees = 4

# Prints "6.0"

print(coffee\_price \* number\_of\_coffees)

# Prints "1.5"

print(coffee\_price)

# Prints "4"

print(number\_of\_coffees)

# Updating the price

coffee\_price = 2.00

# Prints "8.0"

print(coffee\_price \* number\_of\_coffees)

# Prints "2.0"

print(coffee\_price)

# Prints "4"

print(number\_of\_coffees)

We create two variables and assign numeric values to them. Then we perform a calculation on them. This doesn't update the variables! When we update the coffee\_price variable and perform the calculations again, they use the updated values for the variable!

**1.**

You've decided to get into quilting! To calculate the number of squares you'll need for your first quilt let's create two variables: quilt\_width and quilt\_length. Let's make this first quilt 8 squares wide and 12 squares long. Print out the number of squares you'll need to create the quilt!

**2.**

It turns out that quilt required a little more material than you have on hand! Let's only make the quilt 8 squares long. How many squares will you need for this quilt instead?

# First we create the variables

quilt\_width = 8

quilt\_length = 12

# Then we print the size

print(quilt\_width \* quilt\_length)

# Whoops! That's a little too large

quilt\_length = 8

# Let's see how large it is now

print(quilt\_width \* quilt\_length)

# Exponents

Python can also perform exponentiation. In written math, you might see an exponent as a superscript number, but typing superscript numbers isn't always easy on modern keyboards. Since this operation is so related to multiplication, we use the notation \*\*.

# 2 to the 10th power, or 1024

print(2 \*\* 10)

# 8 squared, or 64

print(8 \*\* 2)

# 9 \* 9 \* 9, 9 cubed, or 729

print(9 \*\* 3)

# We can even perform fractional exponents

# 4 to the half power, or 2

print(4 \*\* 0.5)

Here, we compute some simple exponents. We calculate 2 to the 10th power, 8 to the 2nd power, 9 to the 3rd power, and 4 to the 0.5th power.

**1.**

You really like how the square quilts from last exercise came out, and decide that all quilts that you make will be square from now on.

Using the exponent operator, print out how many squares you'll need for a 6x6 quilt, a

7x7 quilt, and an 8x8 quilt.

**2.**

Your 6x6 quilts have taken off so well, 6 people have each requested 6 quilts. Print out how many tiles you would need to make 6 quilts apiece for 6 people.

print(6 \*\* 2)

print(7 \*\* 2)

print(8 \*\* 2)

print(6 \* 6 \* 6 \* 6)

# Modulo

Python offers a companion to the division operator called the modulo operator. The modulo operator is indicated by % and gives the remainder of a division calculation. If the number is divisible, then the result of the modulo operator will be 0.

# Prints 4 because 29 / 5 is 5 with a remainder of 4

print(29 % 5)

# Prints 2 because 32 / 3 is 10 with a remainder of 2

print(32 % 3)

# Modulo by 2 returns 0 for even numbers and 1 for odd numbers

# Prints 0

print(44 % 2)

Here, we use the modulo operator to find the remainder of division operations. We see that 29 % 5 equals 4, 32 % 3 equals 2, and 44 % 2 equals 0.

The modulo operator is useful in programming when we want to perform an action every nth-time the code is run. Can the result of a modulo operation be larger than the divisor? Why or why not?

**1.**

You're trying to divide 16 people into 4 teams. The teams are Team 1, Team 2, Team 3, and Team 0. All of the people line up and count off from person 1 to person 16. Assign their teams by performing a modulo by 4 on each person's number.

In the program, we've created person1\_team and person2\_team. Assign the rest of the players to teams by taking their number and performing a modulo by 4.

person1\_team = 1 % 4

person2\_team = 2 % 4

# Add the rest of the teams here

person3\_team = 3 % 4

person4\_team = 4 % 4

person5\_team = 5 % 4

person6\_team = 6 % 4

person7\_team = 7 % 4

person8\_team = 8 % 4

person9\_team = 9 % 4

person10\_team = 10 % 4

person11\_team = 11 % 4

person12\_team = 12 % 4

person13\_team = 13 % 4

person14\_team = 14 % 4

person15\_team = 15 % 4

person16\_team = 16 % 4

# And here we'll print them out

print("Person 1 belongs to team:", person1\_team)

print("Person 2 belongs to team:", person2\_team)

print("Person 3 belongs to team:", person3\_team)

print("Person 4 belongs to team:", person4\_team)

print("Person 5 belongs to team:", person5\_team)

print("Person 6 belongs to team:", person6\_team)

print("Person 7 belongs to team:", person7\_team)

print("Person 8 belongs to team:", person8\_team)

print("Person 9 belongs to team:", person9\_team)

print("Person 10 belongs to team:", person10\_team)

print("Person 11 belongs to team:", person11\_team)

print("Person 12 belongs to team:", person12\_team)

print("Person 13 belongs to team:", person13\_team)

print("Person 14 belongs to team:", person14\_team)

print("Person 15 belongs to team:", person15\_team)

print("Person 16 belongs to team:", person16\_team)

# Concatenation

The + operator doesn't just add two numbers, it can also "add" two strings! The process of combining two strings is called string concatenation. Performing string concatenation creates a brand-new string comprised of the first string's contents followed by the second string's contents (without any added space in-between).

greeting\_text = "Hey there!"

question\_text = "How are you doing?"

full\_text = greeting\_text + question\_text

# Prints "Hey there!How are you doing?"

print(full\_text)

In this sample of code, we create two variables that hold strings and then concatenate them. But we notice that the result was missing a space between the two, let's add the space in-between using the same concatenation operator!

full\_text = greeting\_text + " " + question\_text

# Prints "Hey there! How are you doing?"

print(full\_text)

Now the code prints the message we expected.

If you want to concatenate a string with a number you will need to make the number a string first, using the str() function. If you're trying to print() a numeric variable you can use commas to pass it as a different argument rather than converting it to a string.

birthday\_string = "I am "

age = 10

birthday\_string\_2 = " years old today!"

# Concatenating an integer with strings is possible if we turn the integer into a string first

full\_birthday\_string = birthday\_string + str(age) + birthday\_string\_2

# Prints "I am 10 years old today!"

print(full\_birthday\_string)

# If we just want to print an integer

# we can pass a variable as an argument to

# print() regardless of whether

# it is a string.

# This also prints "I am 10 years old today!"

print(birthday\_string, age, birthday\_string\_2)

Using str() we can convert variables that are not strings to strings and then concatenate them. But we don't need to convert a number to a string for it to be an argument to a print statement.

Instructions

**1.**

Concatenate the strings and save the message they form in the variable message.

Answer:

string1 = "The wind, "

string2 = "which had hitherto carried us along with amazing rapidity, "

string3 = "sank at sunset to a light breeze; "

string4 = "the soft air just ruffled the water and "

string5 = "caused a pleasant motion among the trees as we approached the shore, "

string6 = "from which it wafted the most delightful scent of flowers and hay."

message = string1 + string2 + string3 + string4 + string5 + string6

print(message)

# Plus Equals

Python offers a shorthand for updating variables. When you have a number saved in a variable and want to add to the number and save the new value over the current variable, you can use the += (plus-equals) operator.

# First we have a variable with a number saved

number\_of\_miles\_hiked = 12

# Then we need to update that variable

# Let's say we hike another two miles today

number\_of\_miles\_hiked += 2

# The new value is the old value

# Plus the number after the plus-equals

print(number\_of\_miles\_hiked)

# Prints 14

Above, we keep a running count of the number of miles a person has gone hiking over time. Instead of recalculating from the start, we keep a grand total and update it when we've gone hiking further.

The plus-equals operator also can be used for string concatenation, like so:

hike\_caption = "What an amazing time to walk through nature!"

# Almost forgot the hashtags!

hike\_caption += " #nofilter"

hike\_caption += " #blessed"

We create the social media caption for the photograph of nature we took on our hike, but then update the caption to include important social media tags we almost forgot.

**1.**

We're doing a little bit of online shopping and find a pair of new sneakers. Right before we check out, we spot a nice sweater and some fun books we also want to purchase!

Update the total\_price to include the prices of nice\_sweater and fun\_books.

total\_price = 0

new\_sneakers = 50.00

total\_price += new\_sneakers

nice\_sweater = 39.00

fun\_books = 20.00

total\_price += nice\_sweater

total\_price += fun\_books

print(total\_price)

# Multi-line Strings

Python strings are very flexible, but if we try to create a string that occupies multiple lines we find ourselves face-to-face with a SyntaxError. Python offers a solution: multi-line strings. By using three quote-marks (""" or ''') instead of one, we tell the program that the string doesn't end until the next triple-quote. This method is useful if the string being defined contains a lot of quotation marks and we want to be sure we don't close it prematurely.

leaves\_of\_grass = """

Poets to come! orators, singers, musicians to come!

Not to-day is to justify me and answer what I am for,

But you, a new brood, native, athletic, continental, greater than

before known,

Arouse! for you must justify me.

"""

In the above example, we assign a famous poet's words to a variable. Even though the quote contains multiple line breaks, the code works!

If a multi-line string isn't assigned a variable or used in an expression it is treated as a comment.

Instructions

**1.**

Assign the string

Stranger, if you passing meet me and desire to speak to me, why

should you not speak to me?

And why should I not speak to you?

to the variable to\_you.

to\_you = """Stranger, if you passing meet me and desire to speak to me, why

should you not speak to me?

And why should I not speak to you?"""

# Review

In this lesson, we accomplished a lot of things! We instructed our computers to print messages, we stored these messages as variables, and we learned to update those messages depending on the part of the program we were in. We performed mathematical calculations and explored some of the mathematical expressions that Python offers us. We learned about errors and other valuable skills that will continue to serve us as we develop our programming skills.

Good job!

Instructions

**1.**

Create variables my\_age, half\_my\_age, greeting, name, and greeting\_with\_name. Assign values to each using your knowledge of division and concatenation!

my\_age = 49

half\_my\_age = (my\_age / 2)

greeting = "Hello my name is "

name = "Jim"

greeting\_with\_name = greeting + name

print (greeting\_with\_name)

# Week 1 Project

## Furniture Store

In this project, you will create variables that store descriptions and prices of various items in a furniture store. You will use what you've learned about creating and updating variables to develop itemizations and receipts for your furniture store's customers.

If you get stuck or confused, remember that your Slack community is there to help!

This project is not graded, and you do not need to submit it anywhere. If you would like to check your results, the [solution code can be found here.](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/On-platform+solutions/furniture_store.py)

# Create Purchasing Information and Receipts for Lovely Loveseats

We've decided to pursue the dream of small-business ownership and open up a furniture store called Lovely Loveseats for Neat Suites on Fleet Street. With our newfound knowledge of Python programming, we're going to build a system to help speed up the process of creating receipts for your customers.

In this project, we will be storing the names and prices of a furniture store's catalog in variables. You will then process the total price and item list of customers, printing them to the output terminal.

Tasks

32/32Complete

Mark the tasks as complete by checking them off

### Adding In The Catalog

1.

Let's add in our first item, the Lovely Loveseat that is the store's namesake. Create a variable called lovely\_loveseat\_description and assign to it the following string:

Lovely Loveseat. Tufted polyester blend on wood. 32 inches high x 40 inches wide x 30 inches deep. Red or white.

Recall string assignment syntax:

variable\_name = "This is a string"

2.

Great, now let's create a price for the loveseat. Create a variable lovely\_loveseat\_price and set it equal to 254.00.

Assign the variable same as the string before, but leave out the quotes since we're storing a number:

numeric\_variable = 10.00

3.

Let's extend our inventory with another characteristic piece of furniture! Create a variable called stylish\_settee\_description and assign to it the following string:

Stylish Settee. Faux leather on birch. 29.50 inches high x 54.75 inches wide x 28 inches deep. Black.

4.

Now let's set the price for our Stylish Settee. Create a variable stylish\_settee\_price and assign it the value of 180.50.

5.

Fantastic, we just need one more item before we're ready for business. Create a new variable called luxurious\_lamp\_description and assign it the following:

Luxurious Lamp. Glass and iron. 36 inches tall. Brown with cream shade.

6.

Let's set the price for this item. Create a variable called luxurious\_lamp\_price and set it equal to 52.15.

7.

In order to be a business, we should also be calculating sales tax. Let's store that in a variable as well.

Define the variable sales\_tax and set it equal to .088. That's 8.8%.

### Our First Customer

8.

Our first customer is making their purchase! Let's keep a running tally of their expenses by defining a variable called customer\_one\_total. Since they haven't purchased anything yet, let's set that variable equal to 0 for now.

9.

We should also keep a list of the descriptions of things they're purchasing. Create a variable called customer\_one\_itemization and set that equal to the empty string "". We'll tack on the descriptions to this as they make their purchases.

An empty string is a string without any characters in it. Assign it just like any other variable:

empty\_string = ""

10.

Our customer has decided they are going to purchase our Lovely Loveseat! Add the price to customer\_one\_total.

The price for the loveseat is saved in lovely\_loveseat\_price. Add it to the total like so:

customer\_one\_total += lovely\_loveseat\_price

11.

Let's start keeping track of the items our customer purchased. Add the description of the Lovely Loveseat to customer\_one\_itemization.

The description is in the lovely\_loveseat\_description variable. Recall you can use the plus-equals operator to add strings to an existing variable.

customer\_one\_itemization += lovely\_loveseat\_description

12.

Our customer has also decided to purchase the Luxurious Lamp! Let's add the price to the customer's total.

Add luxurious\_lamp\_price to customer\_one\_total.

13.

Let's keep the itemization up-to-date and add the description of the Luxurious Lamp to our itemization.

14.

They're ready to check out! Let's begin by calculating sales tax. Create a variable called customer\_one\_tax and set it equal to customer\_one\_total times sales\_tax.

Multiplication uses the \* operator.

customer\_one\_tax = customer\_one\_total \* sales\_tax

15.

Add the sales tax to the customer's total cost.

Update the variable customer\_one\_total by adding customer\_one\_tax to it.

16.

Let's start printing up their receipt! Begin by printing out the heading for their itemization. Print the phrase "Customer One Items:".

Use the print() function and pass in the string to print, like so:

print("Your phrase here")

17.

Print customer\_one\_itemization.

Use the print() function to print the variable like so:

print(customer\_one\_itemization)

18.

Now add a heading for their total cost: print() out "Customer One Total:"

19.

Now print out their total! Our first customer now has a receipt for the things they purchased.

### Our Second Customer

20.

We have a new customer! Let's create their total in a new variable called customer\_two\_total and set it equal to 0.

21.

Begin their itemization with a new variable called customer\_two\_itemization and set that equal to an empty string("").

22.

Our second customer has decided to purchase our Stylish Settee. Add the price of the settee to their total.

Remember the cost of the settee is saved in stylish\_settee\_price. Add this to your total by using the += operator on customer\_two\_total.

23.

Add the description of the Stylish Settee to the itemization.

Concatenate stylish\_settee\_description to customer\_two\_itemization using the += operator.

24.

Customer two has also decided to purchase our Luxurious Lamp (our best-selling item!). Add the price of the Luxurious Lamp to customer\_two\_total.

25.

Update customer\_two\_itemization to include the description of the Luxurious Lamp.

26.

Our customer is ready to check out. Let's calculate the sales tax for their purchase! Save the results in customer\_two\_tax.

Create the variable customer\_two\_tax to be equal to customer\_two\_total times sales\_tax like so:

customer\_two\_tax = customer\_two\_total \* sales\_tax

27.

Add the calculated tax to the total.

28.

Let's print out the second customer's receipt. Start with the heading "Customer 2 Items:"

29.

Now print out customer\_two\_itemization.

30.

Now add a heading for the second customer's total cost. Print "Customer Two Total:".

31.

Print out their total that we've stored in customer\_two\_total.

32.

Congratulations! We created our catalog and served our first two customers. We used our knowledge of strings and numbers to create and update variables. We were able to print out an itemized list and a total cost for each of our customers. Lovely!

Lovely Loveseats for Neat Suites on Fleet Street keeps very precise records of payment:

Customer 1 should pay 333.09119999999996. Customer 2 should pay 253.1232.

My Code:

lovely\_loveseat\_description = """Lovely Loveseat. Tufted polyester blend on wood. 32 inches high x 40 inches wide x 30 inches deep. Red or white."""

lovely\_loveseat\_price = 254.00

stylish\_settee\_description = """Stylish Settee. Faux leather on birch. 29.50 inches high x 54.75 inches wide x 28 inches deep. Black."""

stylish\_settee\_price = 180.50

luxurious\_lamp\_description = """Luxurious Lamp. Glass and iron. 36 inches tall. Brown with cream shade."""

luxurious\_lamp\_price = 52.15

# Sales will be calculated at 8.8%

sales\_tax = .088

#Customer One

customer\_one\_total = 0

customer\_one\_itemization = ""

customer\_one\_total += lovely\_loveseat\_price

customer\_one\_itemization += lovely\_loveseat\_description

customer\_one\_total += luxurious\_lamp\_price

customer\_one\_itemization += luxurious\_lamp\_description

customer\_one\_tax = customer\_one\_total \* sales\_tax

customer\_one\_total += customer\_one\_tax

#Printing receipt

print ("Customer One Items")

print(customer\_one\_itemization)

print(customer\_one\_total)

#Second Customer

customer\_two\_total = 0

customer\_two\_itemization = ""

customer\_two\_total += stylish\_settee\_price

customer\_two\_itemization += stylish\_settee\_description

customer\_two\_total += luxurious\_lamp\_price

customer\_two\_itemization += lovely\_loveseat\_description

customer\_two\_tax = customer\_two\_total \* sales\_tax

customer\_two\_total += customer\_two\_tax

print ("Customer 2 Items")

print (customer\_two\_itemization)

print ("Customer Two Total")

print (customer\_two\_total)