# Introduction to Strings

Welcome to Learn Python: Introduction to Strings!

You've seen and used strings before, but in this lesson we'll take a deep dive into what strings are and how you can use them in your programs. The lesson covers how to manipulate and utilize strings and their properties. You'll also learn how to iterate and loop through strings and use strings effectively in conditional statements.

# Introduction

Words and sentences are fundamental to how we communicate, so it follows that we'd want our computers to be able to work with words and sentences as well.

In Python, the way we store something like a word, a sentence, or even a whole paragraph is as a **string**. A string is a sequence of characters. It can be any length and can contain any letters, numbers, symbols, and spaces.

In this lesson, you will learn more about strings and how they are treated in Python. You will learn how to slice strings, select specific characters from strings, search strings for characters, iterate through strings, and use strings in conditional statements.

Let's get started.

**1.**

Save your favorite word as a string to the variable favorite\_word.

**2.**

Print favorite\_word.

**Solution:**

favorite\_word = ('favorite word')

print(favorite\_word)

# They're all Lists!

A string can be thought of as a **list** of characters.

Like any other list, each character in a string has an index. Consider the string

favorite\_fruit = "blueberry"

We can select specific letters from this string using the index. Let's look at the first letter of the string.

>>> favorite\_fruit[1]

'l'

Whoops, is that the first letter you expected? Notice that the letter at index 1 of "blueberry" isn't b, it's l. This is because the indices of a string start at 0. b is located at the zero index and we could select it using:

>>> favorite\_fruit[0]

'b'

It's important to note that indices of strings must be integers. If you were to try to select a non-integer index we would get a TypeError:

>>> favorite\_fruit[1.5]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: string indices must be integers, not float

Instructions

**1.**

One of the most common things that are represented by strings are names. Save your name as a string to the variable my\_name.

**2.**

Select the first letter of the variable my\_name and save it to first\_initial.

**Solution:**

my\_name=('Jim')

first\_initial=my\_name[0]

# Cut Me a Slice of String

Not only can we select a single character from a string, we can select entire chunks of characters from a string. We can do this with the following syntax:

string\_name[first\_index:last\_index]

This is called slicing a string. When we slice a string, we are creating a **new** string that starts at (and includes) the first\_index and ends at (but excludes) the last\_index. Let's look at some examples of this. Recall our favorite fruit:

>>> favorite\_fruit

'blueberry'

The indices of this string are shown in the diagram below.

Let's say we wanted a new string that contains the letters eberr. We could slice favorite\_fruit as follows:

>>> favorite\_fruit[3:8]

'eberr'

Notice how the character at the first index, e, is INCLUDED, but the character at the last index, y, is EXCLUDED. If you look for the indices 3 and 8 in the diagram, you can see how the y is outside that range.

We can also have open-ended selections. If we remove the first index, the slice starts at the beginning of the string and if we remove the second index the slice continues to the end of the string.

>>> favorite\_fruit[:4]

'blue'

>>> favorite\_fruit[4:]

'berry'

Again, notice how the b from berry is excluded from the first example and included in the second example.

**1.**

You're a programmer working for Copeland's Corporate Company. At this company, each employee's user name is generated by taking the first five letters of their last name.

A new employee, Rodrigo Villanueva, is starting today and you need to create his account. His first\_name and last\_name are stored as strings in **script.py**.

Create a variable new\_account by slicing the first five letters of his last\_name.

**2.**

Temporary passwords for new employees are also generated from their last names.

Create a variable called temp\_password by creating a slice out of the third

through sixth letters of his last\_name.

Hint:

Remember, because indices start at 0, to get the third through sixth characters of a string you would want to use

string\_name[2:6]

**Solution:**

first\_name = "Rodrigo"

last\_name = "Villanueva"

new\_account = last\_name[:5]

temp\_password = last\_name[2:6]

# Concatenating Strings

You can also concatenate two existing strings together into a new string. Consider the following two strings.

fruit\_prefix = "blue"

fruit\_suffix = "berries"

We can create a new string by concatenating them together as follows:

favorite\_fruit = fruit\_prefix + fruit\_suffix

>>> favorite\_fruit

'blueberries'

Notice that there are no spaces added here. You have to manually add in the spaces when concatenating strings if you want to include them.

fruit\_sentence = "My favorite fruit is " + favorite\_fruit

>>> fruit\_sentence

'My favorite fruit is blueberries.`

**1.**

Copeland's Corporate Company has realized that their policy of using the first five letters of an employee's last name as a user name isn't ideal when they have multiple employees with the same last name.

Write a function called account\_generator that takes two inputs, first\_name and last\_name and concatenates the first three letters of each and then returns the new account name.

Hint:

Remember, you can define a new function using

def function\_name(parameter):

**2.**

Test your function on the first\_name and last\_name provided in **script.py** and save it to the variable new\_account

**Solution:**

first\_name = "Julie"

last\_name = "Blevins"

def account\_generator(first\_name, last\_name):

account\_name = first\_name[:3] + last\_name[:3]

return account\_name

new\_account = account\_generator(first\_name, last\_name)

print(new\_account)

# More and More String Slicing (How Long is that String?)

Python comes with some built-in functions for working with strings. One of the most commonly used of these functions is len(). len() returns the number of characters in a string

>>> favorite\_fruit = "blueberry"

>>> len(favorite\_fruit)

9

If you are taking the length of a sentence the spaces are counted as well.

>>> fruit\_sentence = "I love blueberries"

>>> len(fruit\_sentence)

18

len() comes in handy when we are trying to select the last character in a string. You can try to run the following code:

>>> length = len(favorite\_fruit)

>>> favorite\_fruit[length]

But this code would generate an IndexError because, remember, the indices start at 0, so the final character in a string has the index of length(string\_name) - 1.

>>> favorite\_fruit[length-1]

'y'

You could also slice the last several characters of a string using len():

>>> favorite\_fruit[length-4:]

'erry'

Using a len() statement as the starting index and omitting the final index lets you slice n characters from the end of a string where n is the amount you subtract from len().

**1.**

Copeland's Corporate Company also wants to update how they generate temporary passwords for new employees.

Write a function called password\_generator that takes two inputs, first\_name and last\_name and then concatenate the last three letters of each and returns them as a string.

**Hint:**

If we wanted to find the last three letters of a string, we could use len:

last\_three\_letters = string\_name[len(string\_name)-3:]

**2.**

Test your function on the provided first\_name and last\_name and save it to the variable temp\_password.

**Solution:**

first\_name = "Reiko"

last\_name = "Matsuki"

def password\_generator(first\_name, last\_name):

temp\_password = first\_name[len(first\_name)-3:] + last\_name[len(last\_name)-3:]

return temp\_password

temp\_password = password\_generator(first\_name, last\_name)

# Negative Indices

In the previous exercise, we used len() to get a slice of characters at the end of a string.

There's a much easier way to do this, we can use negative indices! Negative indices count backward from the end of the string, so string\_name[-1] is the last character of the string, string\_name[-2] is the second last character of the string, etc.

Here are some examples:

>>> favorite\_fruit = 'blueberry`

>>> favorite\_fruit[-1]

'y'

>>> favorite\_fruit[-2]

'r'

>>> favorite\_fruit[-3:]

'rry'

Notice that we are able to slice the last three characters of 'blueberry' by having a starting index of -3 and omitting a final index.

Instructions

**1.**

Use negative indices to find the the second to last character in company\_motto. Save this to the variable second\_to\_last.

**2.**

Use negative indices to create a slice of the last 4 characters in company\_motto. Save this to the variable final\_word.

**Solution:**

company\_motto = "Copeland's Corporate Company helps you capably cope with the constant cacophony of daily life"

second\_to\_last = company\_motto[-2]

final\_word = company\_motto[-4:]

# Strings are Immutable

So far in this lesson, we've been selecting characters from strings, slicing strings, and concatenating strings. Each time we perform one of these operations we are creating an entirely new string.

This is because strings are immutable. This means that we cannot change a string once it is created. We can use it to create other strings, but we cannot change the string itself.

This property, generally, is known as mutability. Data types that are mutable can be changed, and data types, like strings, that are immutable cannot be changed.

**1.**

The most recent hire at Copeland's Corporate Company is a fellow named Rob Daily. Unfortunately, Human Resources seem to have made a bit of a typo and sent over the wrong first\_name.

Try changing the first character of first\_name by running

first\_name[0] = "R"

**2.**

Oh right! Strings are immutable, so we can't change an individual character. Okay, that's no problem we can still fix this!

Concatenate the string "R" with a slice of first\_name that includes everything but the first character and save it to a new string fixed\_first\_name.

**Solution:**

first\_name = "Bob"

last\_name = "Daily"

# first\_name[0] = "R"

fixed\_first\_name = "R" + first\_name[-2:]

print(fixed\_first\_name)

# Escape Characters

Occasionally when working with strings, you'll find that you want to include characters that already have a special meaning in python. For example let's say I create the string

favorite\_fruit\_conversation = "He said, "blueberries are my favorite!""

We'll have accidentally ended the string before we wanted to by including the " character. The way we can do this is by introducing escape characters. By adding a backslash in front of the special character we want to escape, \", we can include it in a string.

favorite\_fruit\_conversation = "He said, \"blueberries are my favorite!\""

Now it works!

**1.**

When Rob Daily (remember him? From the last exercise?) set up his account he set his password to be

theycallme"crazy"91

His password was causing some errors in the system because of the " marks. Rewrite his password using escape characters and save it to the variable password.

**Solution:**

password = "theycallme\"crazy\"91"

print(password)

# Iterating through Strings

Now you know enough about strings that we can start doing the really fun stuff!

Because strings are lists, that means we can iterate through a string using for or while loops. This opens up a whole range of possibilities of ways we can manipulate and analyze strings. Let's take a look at an example.

def print\_each\_letter(word):

for letter in word:

print(letter)

This code will iterate through each letter in a given word and will print it to the terminal.

>>> favorite\_color = "blue"

>>> print\_each\_letter(favorite\_color)

'b'

'l'

'u'

'e'

Let's try a couple problems where we need to iterate through a string.

**1.**

Let's replicate a function you are already familiar with, len().

Write a new function called get\_length() that takes a string as an input and returns the number of characters in that string. Do this by iterating through the string, don't cheat and use len()!

**Hint:**

Using a counter variable and a for loop is a great way to count things. For example look at the following code:

counter = 0

for something in something\_else:

counter += 1

**Solution:**

def get\_length(word):

counter = 0

for letter in word:

counter += 1

return counter

# Strings and Conditionals (Part One)

Now that we are iterating through strings, we can really explore the potential of strings. When we iterate through a string we do something with each character. By including conditional statements inside of these iterations, we can start to do some really cool stuff.

Take a look at the following code:

favorite\_fruit = "blueberry"

counter = 0

for character in favorite\_fruit:

if character == "b":

counter = counter + 1

print(counter)

This code will count the number of bs in the string "blueberry" (hint: it's two). Let's take a moment and break down what exactly this code is doing.

First, we define our string, favorite\_fruit, and a variable called counter, which we set equal to zero. Then the for loop will iterate through each character in favorite\_fruit and compare it to the letter b.

Each time a character equals b the code will increase the variable counter by one. Then, once all characters have been checked, the code will print the counter, telling us how many bs were in "blueberry". This is a great example of how iterating through a string can be used to solve a specific application, in this case counting a certain letter in a word.

**1.**

Write a function called letter\_check that takes two inputs, word and letter.

This function should return True if the word contains the letter and False if it does not.

Hint:

Make sure to test your function!

Try running

letter\_check("strawberry", "a")

and

letter\_check("strawberry", "o")

What do you expect the function to return when you run these? What does it return?

**Solution:**

def letter\_check(word, letter):

for character in word:

if character == letter:

return True

return False

letter\_check("strawberry", "a")

letter\_check("strawberry", "o")

# Strings and Conditionals (Part Two)

There's an even easier way than iterating through the entire string to determine if a character is in a string. We can do this type of check more efficiently using in. in checks if one string is part of another string.

Here is what the syntax of in looks like:

letter in word

Here, letter in word is a boolean expression that is True if the string letter is in the string word. Here are some examples:

>>>"e" in "blueberry"

True

>>> "a" in "blueberry"

False

In fact, this method is more powerful than the function you wrote in the last exercise because it works not only with letters, but with entire strings as well.

>>> "blue" in "blueberry"

True

>>> "blue" in "strawberry"

False

**1.**

Write a function called contains that takes two arguments, big\_string and little\_string and returns True if big\_string contains little\_string.

For example contains("watermelon", "melon") should return True and contains("watermelon", "berry") should return False.

**2.**

Write a function called common\_letters that takes two arguments, string\_one and string\_two and then returns a list with all of the letters they have in common.

The letters in the returned list should be unique. For example,

common\_letters("banana", "cream")

should return ['a'].

Hint:

.append() will be useful in adding the shared letters to the list you will eventually return.

Also, make sure to test your function!

**Solution:**

def contains(big\_string, little\_string):

return little\_string in big\_string

def common\_letters(string\_one, string\_two):

common = []

for letter in string\_one:

if (letter in string\_two) and not (letter in common):

common.append(letter)

return common

# Review

Great work! I hope you are now starting to see the potential of strings and how they can be used to solve a huge variety of problems.

In this lesson you learned:

* A string is a list of characters.
* A character can be selected from a string using its index string\_name[index]. These indices start at 0.
* A 'slice' can be selected from a string. These can be between two indices or can be open-ended, selecting all of the string from a point.
* Strings can be concatenated to make larger strings.
* len() can be used to determine the number of characters in a string.
* Strings can be iterated through using for loops.
* Iterating through strings opens up a huge potential for applications, especially when combined with conditional statements.

Let's put your new skills to the test!

**1.**

Copeland's Corporate Company has finalized what they want to their username and temporary password creation to be and have enlisted your help, once again, to build the function to generate them. In this exercise, you will create two functions, username\_generator and password\_generator.

Let's start with username\_generator. Create a function called username\_generator take two inputs, first\_name and last\_name and returns a username. The username should be a slice of the first three letters of their first name and the first four letters of their last name. If their first name is less than three letters or their last name is less than four letters it should use their entire names.

For example, if the employee's name is Abe Simpson the function should generate the username AbeSimp.

**2.**

Great work! Now for the temporary password, they want the function to take the input user name and shift all of the letters by one to the right, so the last letter of the username ends up as the first letter and so forth. For example, if the username is AbeSimp, then the temporary password generated should be pAbeSim.

Start by defining the function password\_generator so that it takes one input, username and in it define an empty string named password.

**3.**

Inside password\_generator create a for loop that iterates through the indices username by going from 0 to len(username).

To shift the letters right, at each letter the for loop should add the previous letter to the string password.

**Hint:**

Remember, you can use range(x,y) to generate a list of values between and including x and y. This is how you should iterate through the username.

**Solution:**

def username\_generator(first\_name, last\_name):

if len(first\_name) < 3:

user\_name = first\_name

else:

user\_name = first\_name[0:3]

if len(last\_name) < 4:

user\_name += last\_name

else:

user\_name += last\_name[0:4]

return user\_name

def password\_generator(user\_name):

password = ""

for i in range(0, len(user\_name)):

password += user\_name[i-1]

return password

**Introduction**

Do you have a gigantic string that you need to parse for information? Do you need to sanitize a users input to work in a function? Do you need to be able to generate outputs with variable values? All of these things can be accomplished with *string methods*!

Python comes with built-in *string methods* that gives you the power to perform complicated tasks on strings very quickly and efficiently. These string methods allow you to change the case of a string, split a string into many smaller strings, join many small strings together into a larger string, and allow you to neatly combine changing variables with string outputs.

In the previous lesson, you worked len(), which was a *function* that determined the number of characters in a string. This, while similar, was NOT a string method. String methods all have the same syntax:

string\_name.string\_method(arguments)

Unlike len(), which is called with a string as it's argument, a string method is called at the end of a string and each one has its own method specific arguments.

Instructions

The diagram shows all of the string methods you can expect to learn in this lesson. Take a quick look at them and then let's get started!

**Formatting Methods**

There are three string methods that can change the casing of a string. These are .lower(), .upper(), and .title().

* .lower() returns the string with all lowercase characters.
* .upper() returns the string with all uppercase characters.
* .title() returns the string in title case, which means the first letter of each word is capitalized.

Here's an example of .lower() in action:

>>> favorite\_song = 'SmOoTH'

>>> favorite\_song\_lowercase = favorite\_song.lower()

>>> favorite\_song\_lowercase

'smooth'

Every character was changed to lowercase! It's important to remember that string methods can only **create** new strings, they do not change the original string.

>>> favorite\_song

'SmOoTH'

See, it's still the same! These string methods are great for sanitizing user input and standardizing the formatting of your strings.

**1.**

You're a programmer working for an organization that is trying to digitize and store poetry called *Preserve the Verse*.

You've been given two strings, the title of a poem and it's author, and have been asked to reformat them slightly to fit the conventions of the organization's database.

Make poem\_title have title case and save it to poem\_title\_fixed.

**2.**

Print poem\_title and poem\_title\_fixed.

How did the string change?

**3.**

The organization's database also needs the author's name to be uppercase only.

Make poem\_author uppercase and save it to poem\_author\_fixed.

**4.**

Print poem\_author and poem\_author\_fixed.

Again, how did the string change?

**Solution:**

poem\_title = "spring storm"

poem\_author = "William Carlos Williams"

poem\_title\_fixed = poem\_title.title()

print(poem\_title)

print(poem\_title\_fixed)

poem\_author\_fixed = poem\_author.upper()

print(poem\_author)

print(poem\_author\_fixed)

# Splitting Strings

.upper(), .lower(), and .title() all are performed on an existing string and produce a string in return. Let's take a look at a string method that returns a different object entirely!

.split() is performed on a string, takes one argument, and returns a list of substrings found between the given argument (which in the case of .split() is known as the delimiter). The following syntax should be used:

string\_name.split(delimiter)

If you do not provide an argument for .split() it will default to splitting at spaces.

For example, consider the following strings:

>>> man\_its\_a\_hot\_one = "Like seven inches from the midday sun"

>>> man\_its\_a\_hot\_one.split()

['Like', 'seven', 'inches', 'from', 'the', 'midday', 'sun']

.split returned a list with each word in the string. Important to note: if we run .split() on a string with no spaces, we will get the same string in return.

**1.**

In the code editor is a string of the first line of the poem Spring Storm by William Carlos Williams.

Use .split() to create a list called line\_one\_words that contains each word in this line of poetry.

**Solution:**

line\_one = "The sky has given over"

line\_one\_words = line\_one.split()

# Splitting Strings II

If we provide an argument for .split() we can dictate the character we want our string to be split on. This argument should be provided as a string itself.

Consider the following example:

>>> greatest\_guitarist = "santana"

>>> greatest\_guitarist.split('n')

['sa', 'ta', 'a']

We provided 'n' as the argument for .split() so our string "santana" got split at each 'n' character into a list of three strings.

What do you think happens if we split the same string at 'a'?

>>> greatest\_guitarist.split('a')

['s', 'nt', 'n', ' ']

Notice that there is an unexpected extra '' string in this list. When you split a string on a character that it also ends with, you'll end up with an empty string at the end of the list.

You can use any string as the argument for .split(), making it a versatile and powerful tool.

**1.**

Your boss at the Poetry organization sent over a bunch of author names that he wants you to prepare for importing into the database. Annoyingly, he sent them over as a long string with the names separated by commas.

Using .split() and the provided string, create a list called author\_names containing each individual author name as it's own string.

**2.**

Great work, but now it turns out they didn't want poet's first names (why didn't they just say that the first time!?)

Create another list called author\_last\_names that only contains the last names of the poets in the provided string.

**Hint:**

There are several ways to do this, but one way is to iterate through the list you created in part one and use .split(), negative indexing, and .append() to construct the new list.

**Solution:**

authors = "Audre Lorde,Gabriela Mistral,Jean Toomer,An Qi,Walt Whitman,Shel Silverstein,Carmen Boullosa,Kamala Suraiyya,Langston Hughes,Adrienne Rich,Nikki Giovanni"

author\_names = authors.split(',')

print(author\_names)

author\_last\_names = []

for name in author\_names:

author\_last\_names.append(name.split()[-1])

print(author\_last\_names)

# Splitting Strings III

We can also split strings using escape sequences. Escape sequences are used to indicate that we want to split by something in a string that is not necessarily a character. The two escape sequences we will cover here are

* \n Newline
* \t Horizontal Tab

Newline or \n will allow us to split a multi-line string by line breaks and \t will allow us to split a string by tabs. \t is particularly useful when dealing with certain datasets because it is not uncommon for data points to be separated by tabs.

Let's take a look at an example of splitting by an escape sequence:

smooth\_chorus = \

"""And if you said, "This life ain't good enough."

I would give my world to lift you up

I could change my life to better suit your mood

Because you're so smooth"""

chorus\_lines = smooth\_chorus.split('\n')

print(chorus\_lines)

This code is splitting the multi-line string at the newlines (\n) which exist at the end of each line and saving it to a new list called chorus\_lines. Then it prints chorus\_lines which will produce the output

['And if you said, "This life ain\'t good enough."', 'I would give my world to lift you up', 'I could change my life to better suit your mood', "Because you're so smooth"]

The new list contains each line of the original string as it's own smaller string. Also, notice that Python automatically escaped the ' character when it created the new list.

**1.**

The organization has sent you over the full text for William Carlos Williams poem Spring Storm. They want you break the poem up into its individual lines.

Create a list called spring\_storm\_lines that contains a string for each line of Spring Storm.

Hint:

You will have to use .split() and the escape character for a newline, /n.

**Solution:**

spring\_storm\_text = \

"""The sky has given over

its bitterness.

Out of the dark change

all day long

rain falls and falls

as if it would never end.

Still the snow keeps

its hold on the ground.

But water, water

from a thousand runnels!

It collects swiftly,

dappled with black

cuts a way for itself

through green ice in the gutters.

Drop after drop it falls

from the withered grass-stems

of the overhanging embankment."""

spring\_storm\_lines = spring\_storm\_text.split('\n')

# Joining Strings

Now that you've learned to break strings apart using .split(), let's learn to put them back together using .join(). .join() is essentially the opposite of .split(), it joins a list of strings together with a given delimiter. The syntax of .join() is:

'delimiter'.join(list\_you\_want\_to\_join)

Now this may seem a little weird, because with .split() the argument was the delimiter, but now the argument is the list. This is because join is still a string method, which means it has to act on a string. The string .join() acts on is the delimiter you want to join with, therefore the list you want to join has to be the argument.

It's can be a bit confusing, so let's take a look at an example.

>>> my\_munequita = ['My', 'Spanish', 'Harlem', 'Mona', 'Lisa']

>>> ' '.join(my\_munequita)

'My Spanish Harlem Mona Lisa'

We take the list of strings, my\_munequita, and we joined it together with our delimiter, ' ', which is a space. The space is important if you are trying to build a sentence from words, otherwise, we would have ended up with:

>>> ''.join(my\_munequita)

'MySpanishHarlemMonaLisa'

**1.**

You've been provided with a list of words from the first line of Jean Toomer's poem [*Reapers*](https://www.poetryfoundation.org/poems/46405/reapers).

Use .join() to combine these words into a sentence and save that sentence as the strin

g reapers\_line\_one.

**Hint:**

Make sure that you are running join on a space, ' ', otherwise you'll mash the words together.

**Solution:**

reapers\_line\_one\_words = ["Black", "reapers", "with", "the", "sound", "of", "steel", "on", "stones"]

reapers\_line\_one = ' '.join(reapers\_line\_one\_words)

# Joining Strings II

In the last exercise, we joined together a list of words using a space as the delimiter to create a sentence. In fact, you can use any string as a delimiter to join together a list of strings. For example, if we have the list

>>> santana\_songs = ['Oye Como Va', 'Smooth', 'Black Magic Woman', 'Samba Pa Ti', 'Maria Maria']

We could join this list together with ANY string. One often used string is a comma , because then we can create a string of comma separated variables, or CSV.

>>> santana\_songs\_csv = ','.join(santana\_songs)

>>> santana\_songs\_csv

'Oye Como Va,Smooth,Black Magic Woman,Samba Pa Ti,Maria Maria'

You'll often find data stored in CSVs because it is an efficient, simple file type used by popular programs like Excel or Google Spreadsheets.

You can also join using escape sequences as the delimiter. Consider the following example:

smooth\_fifth\_verse\_lines = ['Well I\'m from the barrio', 'You hear my rhythm on your radio', 'You feel the turning of the world so soft and slow', 'Turning you \'round and \'round']

smooth\_fifth\_verse = '\n'.join(smooth\_fifth\_verse\_lines)

print(smooth\_fifth\_verse)

This code is taking the list of strings and joining them using a newline \n as the delimiter. Then it prints the result and produces the output:

Well I'm from the barrio

You hear my rhythm on your radio

You feel the turning of the world so soft and slow

Turning you 'round and 'round

**1.**

You've been given a list, winter\_trees\_lines, that contains all the lines to William Carlos Williams poem, Winter Trees. You've been asked to join together the strings in the list together into a single string that can be used to display the full poem. Name this string winter\_trees\_full.

Print your result to the terminal. Make sure that each line of the poem appears on a new line in your string.

**Hint:**

Use \n, the escape character for a line break`.

**Solution:**

winter\_trees\_lines = ['All the complicated details', 'of the attiring and', 'the disattiring are completed!', 'A liquid moon', 'moves gently among', 'the long branches.', 'Thus having prepared their buds', 'against a sure winter', 'the wise trees', 'stand sleeping in the cold.']

winter\_trees\_full = '\n'.join(winter\_trees\_lines)

print(winter\_trees\_full)

# .strip()

When working with strings that come from real data, you will often find that the strings aren't super clean. You'll find lots of extra whitespace, unnecessary linebreaks, and rogue tabs.

Python provides a great method for cleaning strings: .strip(). Stripping a string removes all whitespace characters from the beginning and end. Consider the following example:

>>> featuring = " rob thomas "

>>> featuring.strip()

'rob thomas'

All the whitespace on either side of the string has been stripped, but the whitespace in the middle has been preserved.

You can also use .strip() with a character argument, which will strip that character from either end of the string.

>>> featuring = "!!!rob thomas !!!!!"

>>> featuring.strip('!')

'rob thomas '

By including the argument '!' we are able to strip all of the ! characters from either side of the string. Notice that now that we've included an argument we are no longer stripping whitespace, we are ONLY stripping the argument.

**1.**

They sent over another list containing all the lines to the Audre Lorde poem, Love, Maybe. They want you to join together all of the lines into a single string that can be used to display the poem again, but this time, you've noticed that the list contains a ton of unnecessary whitespace that doesn't appear in the actual poem.

First, use .strip() on each line in the list to remove the unnecessary whitespace and save it as a new list love\_maybe\_lines\_stripped.

**2.**

.join() the lines in love\_maybe\_lines\_stripped together into one large multi-line string, love\_maybe\_full, that can be printed to display the poem.

Each line of the poem should show up on its own line.

**3.**

Print love\_maybe\_full.

**Solution**:

love\_maybe\_lines = ['Always ', ' in the middle of our bloodiest battles ', 'you lay down your arms', ' like flowering mines ','\n' ,' to conquer me home. ']

love\_maybe\_lines\_stripped = []

for line in love\_maybe\_lines:

love\_maybe\_lines\_stripped.append(line.strip())

love\_maybe\_full = '\n'.join(love\_maybe\_lines\_stripped)

print(love\_maybe\_full)

# Replace

The next string method we will cover is .replace(). Replace takes two arguments and replaces all instances of the first argument in a string with the second argument. The syntax is as follows

string\_name.replace(character\_being\_replaced, new\_character)

Great! Let's put it in context and look at an example.

>>> with\_spaces = "You got the kind of loving that can be so smooth"

>>> with\_underscores = with\_spaces.replace(' ', '\_')

>>> with\_underscores

'You\_got\_the\_kind\_of\_loving\_that\_can\_be\_so\_smooth'

Here we used .replace() to change every instance of a space in the string above to be an underscore instead.

**1.**

The poetry organization has sent over the bio for Jean Toomer as it currently exists on their site. Notice that there was a mistake with his last name and all instances of Toomer are lacking one o.

Use .replace() to change all instances of Tomer in the bio to Toomer. Save the updated bio to the string toomer\_bio\_fixed.

**Solution:**

toomer\_bio = \

"""

Nathan Pinchback Tomer, who adopted the name Jean Tomer early in his literary career, was born in Washington, D.C. in 1894. Jean is the son of Nathan Tomer was a mixed-race freedman, born into slavery in 1839 in Chatham County, North Carolina. Jean Tomer is most well known for his first book Cane, which vividly portrays the life of African-Americans in southern farmlands.

"""

toomer\_bio\_fixed = toomer\_bio.replace("Tomer", "Toomer")

# .find()

Another interesting string method is .find(). .find() takes a string as an argument and searching the string it was run on for that string. It then returns the first index value where that string is located.

Here's an example:

>>> 'smooth'.find('t')

'4'

We searched the string smooth for the string t and found that it at the fourth index spot, so .find() returned 4.

You can also search for larger strings, and .find() will return the index value of the first character of that string.

>>>"smooth".find('oo')

'2'

Notice here that 2 is the index of the first o.

**1.**

In the code editor is the first line of Gabriela Mistral's poem [*God Wills It*](https://www.poetryfoundation.org/poetrymagazine/browse?contentId=23104).

At what index place does the word "disown" appear? Save that index place to the variable disown\_placement.

**Solution:**

god\_wills\_it\_line\_one = "The very earth will disown you"

disown\_placement = god\_wills\_it\_line\_one.find('disown')

print(disown\_placement)

# .format()

Python also provides a handy string method for including variables in strings. This method is .format(). .format() takes variables as an argument and includes them in the string that it is run on. You include {} marks as placeholders for where those variables will be imported.

Consider the following function:

def favorite\_song\_statement(song, artist):

return "My favorite song is {} by {}.".format(song, artist)

The function favorite\_song\_statement takes two arguments, song and artist, then returns the a string that includes both of the arguments and prints a sentence. Note: .format() can take as many arguments are there are {} in the string it is run on, which in this case in two.

Here's an example of the function being run:

>>> favorite\_song\_statement("Smooth", "Santana")

"My favorite song is Smooth by Santana"

Now you may be asking yourself, I could have written this function using string concatenation instead of .format(), why is this method better? The answer is legibility and reusability. It is much easier to picture the end result .format() than it is to picture the end result of string concatenation and legibility is everything. You can also reuse the same base string with different variables, allowing you to cut down on unnecessary, hard to interpret code.

**1.**

Write a function called poem\_title\_card that takes two inputs poet and title. The function should use .format() to return the following string:

The poem "[TITLE]" is written by [POET].

For example, if the function is given the inputs

poem\_title\_card("Walt Whitman", "I Hear America Singing")

It should return the string

The poem "I Hear America Singing" is written by Walt Whitman.

Hint:

Remember to escape the " characters!

**Solution**:

def poem\_title\_card(poet, title):

poem\_desc = "The poem \"{}\" is written by {}".format(title, poet)

return poem\_desc

# .format() II

.format() can be made even more legible for other people reading your code by including keywords. Previously with .format(), you had to make sure that your variables appeared as arguments in the same order that you wanted them to appear in the string, which just added unnecessary complications when writing code.

By including keywords in the string and in the arguments, you can remove that ambiguity. Let's look at an example.

def favorite\_song\_statement(song, artist):

return "My favorite song is {song} by {artist}.".format(song=song, artist=artist)

Now it is clear to anyone reading the string what it supposed to return, they don't even need to look at the arguments of .format() in order to get a clear understanding of what is supposed to happen. You can even reverse the order of artist and song in the code above and it will work the same way. This makes writing AND reading the code much easier.

**1.**

The function poem\_description is supposed to use .format() to print out some quick information about a poem, but it seems to be causing some errors currently.

Fix the function by using keywords in the .format() method.

**2.**

Run poem\_description with the following arguments and save the results to the variable my\_beard\_description:

author = "Shel Silverstein"

title = "My Beard"

original\_work = "Where the Sidewalk Ends"

publishing\_date = "1974"

**Solution:**

def poem\_description(publishing\_date, author, title, original\_work):

poem\_desc = "The poem {title} by {author} was originally published in {original\_work} in {publishing\_date}.".format(publishing\_date = publishing\_date, author = author, title = title, original\_work = original\_work)

return poem\_desc

author = "Shel Silverstein"

title = "My Beard"

original\_work = "Where the Sidewalk Ends"

publishing\_date = "1974"

my\_beard\_description = poem\_description(publishing\_date, author, title, original\_work)

print(my\_beard\_description)

# Review

Excellent work! This lesson has shown you the vast variety of string methods and their power. Whatever the problem you are trying to solve, if you are working with strings then string methods are likely going to be part of the solution.

Over this lesson you've learned:

* .upper(), .title(), and .lower() adjust the casing of your string.
* .split() takes a string and creates a list of substrings.
* .join() takes a list of strings and creates a string.
* .strip() cleans off whitespace, or other noise from the beginning and end of a string.
* .replace() replaces all instances of a character/string in a string with another character/string.
* .find() searches a string for a character/string and returns the index value that character/string is found at.
* .format() and f-strings allow you to interpolate a string with variables.

Well I've been stringing you along for long enough, let's get some more practice in!

**1.**

Preserve the Verse has one final task for you. They have delivered you a string that contains a list of poems, titled highlighted\_poems, they want to highlight on the site, but they need your help to parse the string into something that can display the name, title, and publication date of the highlighted poems on the site.

First, start by printing highlighted\_poems to the terminal and see how it displays.

**2.**

The information for each poem is separated by commas, and within this information is the title of the poem, the author, and the date of publication.

Start by splitting highlighted\_poems at the commas and saving it to highlighted\_poems\_list.

Hint:

Recall that the syntax for splitting a string into a list is:

my\_string.split(delimiter)

**3.**

Print highlighted\_poems\_list, how does the structure of the data look now?

**4.**

Notice that there is inconsistent whitespace in highlighted\_poems\_list. Let's clean that up.

Start by creating a new empty list, highlighted\_poems\_stripped.

Then, iterate through highlighted\_poems\_list using a for loop and for each poem strip away the whitespace and append it to your new list, highlighted\_poems\_stripped.

**5.**

Print highlighted\_poems\_stripped.

Looks good! All the whitespace is cleaned up.

**6.**

Next we want to break up all the information for each poem into it's own list, so we end up with a list of lists.

Create a new empty list called highlighted\_poems\_details.

**7.**

Iterate through highlighted\_poems\_stripped and split each string around the : characters and append the new lists into highlighted\_poems\_details.

**8.**

Great! Now we want to separate out all of the titles, the poets, and the publication dates into their own lists.

Create three new empty lists, titles, poets, and dates.

**9.**

Iterate through highlighted\_poems\_details and for each list in the list append the appropriate elements into the lists titles, poets, and dates.

For example, for the poem The Shadow (1915) by William Carlos Williams your code should be adding "The Shadow" to titles, "William Carlos Williams" to poets, and "1915" to dates.

**10.**

Finally, write a for loop that uses either f-strings or .format() to prints out the following string for each poem:

The poem TITLE was published by POET in DATE.

**Solution:**

highlighted\_poems = "Afterimages:Audre Lorde:1997, The Shadow:William Carlos Williams:1915, Ecstasy:Gabriela Mistral:1925, Georgia Dusk:Jean Toomer:1923, Parting Before Daybreak:An Qi:2014, The Untold Want:Walt Whitman:1871, Mr. Grumpledump's Song:Shel Silverstein:2004, Angel Sound Mexico City:Carmen Boullosa:2013, In Love:Kamala Suraiyya:1965, Dream Variations:Langston Hughes:1994, Dreamwood:Adrienne Rich:1987"

# print(highlighted\_poems)

highlighted\_poems\_list = highlighted\_poems.split(',')

# print(highlighted\_poems\_list)

highlighted\_poems\_stripped = []

for poem in highlighted\_poems\_list:

highlighted\_poems\_stripped.append(poem.strip())

# print(highlighted\_poems\_stripped)

highlighted\_poems\_details = []

for poem in highlighted\_poems\_stripped:

highlighted\_poems\_details.append(poem.split(':'))

titles = []

poets = []

dates = []

for poem in highlighted\_poems\_details:

titles.append(poem[0])

poets.append(poem[1])

dates.append(poem[2])

for i in range(0,len(highlighted\_poems\_details)):

print('The poem {} was published by {} in {}'.format(titles[i], poets[i], dates[i]))

# Thread Shed

# Project

About 1 hour, 5 minutes

In this project, you will utilize your knowledge of string methods and lists to sort through a messy CSV that stores all of the daily sales data for the local sewing supply shop, Thread Shed.

Then, using the cleaned up data, you will answer several questions about the daily business dealing of Thread Shed.

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/thread_shed.py)

Good luck and have fun!

# Thread Shed

You've recently been hired as a cashier at the local sewing hobby shop, **Thread Shed**. Some of your daily responsibilities involve tallying the number of sales during the day, calculating the total amount of money made, and keeping track of the names of the customers.

Unfortunately, the **Thread Shed** has an extremely out-dating register system and stores all of the transaction information in one huge unwieldy string called daily\_sales.

All day, for each transaction, the name of the customer, amount spent, types of thread purchased, and the date of sale is all recorded in this same string. Your task is to use your Python skills to iterate through this string and clean up each transaction and store all the information in easier to access lists.

Python: String Methods

# Thread Shed

You've recently been hired as a cashier at the local sewing hobby shop, **Thread Shed**. Some of your daily responsibilities involve tallying the number of sales during the day, calculating the total amount of money made, and keeping track of the names of the customers.

Unfortunately, the **Thread Shed** has an extremely out-dating register system and stores all of the transaction information in one huge unwieldy string called daily\_sales.

All day, for each transaction, the name of the customer, amount spent, types of thread purchased, and the date of sale is all recorded in this same string. Your task is to use your Python skills to iterate through this string and clean up each transaction and store all the information in easier to access lists.

Mark the tasks as complete by checking them off

### Break up `daily\_sales` in easy to understand lists `customers`, `sales`, and `threads\_sold`.

1.

First, take a minute to inspect the string daily\_sales in the code editor.

How is each transaction stored? How is each piece of data within the transaction stored?

Start thinking about how we can split up this string into its individual pieces of data.

2.

It looks like each transaction is separated from the next transaction by a ,, and then each piece of data within a transaction is separated by the artifact ;,;.

If we want to split up daily\_sales into a list of individual transactions, we are going to want to split by ,, but first, we need to replace the artifact ;,; to something without a comma, so we don't split any transactions themselves.

Replace all the instances of ;,; in daily\_sales with some other character and save the result to daily\_sales\_replaced.

For example, if we wanted to replace all instances of "a" in the string my\_string with "b", we can run

my\_string\_replaced = my\_string.replace("a","b")

If you are having trouble thinking of a character to replace ;,; with, try using ;.

3.

Now we can split the string into a list of each individual transaction.

Split daily\_sales\_replaced around commas and save it to a new list daily\_transactions.

Remember, to split a string around a certain delimiter, use the code

new\_list = my\_string.split(delimiter)

4.

Print daily\_transactions.

How does it look?

5.

Our next step is to split each individual transaction into a list of its data points.

First, define an empty list daily\_transactions\_split

6.

Now, iterate through daily\_transactions (remember, this is a list of strings currently), and for each transaction, split the string around whatever character you replaced the ;,; artifacts with in **Step 2**.

Append each of these split strings (which are lists now) to our new list daily\_transactions\_split.

An easy way to perform a task on every item in a list and add it to a new list is by following this format:

new\_list = []

for item in list:

new\_list.append(task(item))

Think about how you can use code of this form to accomplish the given task.

7.

Print daily\_transactions\_split.

How's it looking?

8.

It looks like each data item has inconsistent whitespace around it. First, define an empty list transactions\_clean.

Now, Iterate through daily\_transactions\_split and for each transaction iterate through the different data points and strip off any whitespace.

Add each of these cleaned up transactions to the new list transactions\_clean.

This will require two for loops. One to iterate through the outer list, daily\_transactions\_split and one to iterate through each of the transactions and perform the .strip() on the data points.

9.

Print transactions\_clean.

If you performed the last step correctly, you shouldn't see any unnecessary whitespace.

10.

Create three empty lists. customers, sales, and thread\_sold. We are going to collect the individual data points for each transaction in these lists.

11.

Now, iterate through transactions\_clean and for each transaction:

1. Append the customers name to customers.
2. Append the amount of the sale to sales.
3. Append the threads sold to thread\_sold.

12.

Print customers, sales, and thread\_sold to make sure each list is what you are expected.

### Determine the total value of the days sales.

13.

Now we want to know how much Thread Shed made in a day.

First, define a variable called total\_sales and set it equal to 0.

14.

Now, consider the list sales. It is a list of strings that we want to sum. In order for us to sum these values, we will have to remove the $, and set them equal to floats.

Iterate through sales and for each item, strip off the $, set it equal to a float, and add it to total\_sales

15.

Print total sales.

How much did we make today?

### How much thread of any specific color was sold?

16.

Finally, we want to determine how many of each color thread we sold today. Let's start with a single color, and then we'll generalize it.

First, print out thread\_sold and inspect it.

17.

We see that thread\_sold is a list of strings, some are single colors and some are multiple colors separated by the & character.

The end product we want here is a list that contains an item for each color thread sold, so no strings that have multiple colors in them.

First, define an empty list thread\_sold\_split.

18.

Next, iterate through thread\_sold. For each item, check if it is a single color or multiple colors. If it is a single color, append that color to thread\_sold\_split.

If it is multiple colors, first split the string around the & character and then add each color indivudally to thread\_sold\_split.

19.

Great, now we have a list thread\_sold\_split that contains an entry with the color of every thread sold today.

Define a function called color\_count that takes one argument, color. The function should iterate through thread\_sold\_split and count the number of times the item is equal to argument, color. Then, it should return this count.

20.

Test your new function by running color\_count('white').

Your function should return 28.

21.

Define a list called colors that stores all of the colored threads that Thread Shed offers:

colors = ['red','yellow','green','white','black','blue','purple']

22.

Now, using the list colors, the string method .format(), and the function color\_count, iterate through thread\_sold\_split and print a sentence that says how many threads of each color were sold today.

The output of this function should look like:

Thread Shed sold 28 threads of white thread today.

daily\_sales = \

"""Edith Mcbride ;,;$1.21 ;,; white ;,;

09/15/17 ,Herbert Tran ;,; $7.29;,;

white&blue;,; 09/15/17 ,Paul Clarke ;,;$12.52

;,; white&blue ;,; 09/15/17 ,Lucille Caldwell

;,; $5.13 ;,; white ;,; 09/15/17,

Eduardo George ;,;$20.39;,; white&yellow

;,;09/15/17 , Danny Mclaughlin;,;$30.82;,;

purple ;,;09/15/17 ,Stacy Vargas;,; $1.85 ;,;

purple&yellow ;,;09/15/17, Shaun Brock;,;

$17.98;,;purple&yellow ;,; 09/15/17 ,

Erick Harper ;,;$17.41;,; blue ;,; 09/15/17,

Michelle Howell ;,;$28.59;,; blue;,; 09/15/17 ,

Carroll Boyd;,; $14.51;,; purple&blue ;,;

09/15/17 , Teresa Carter ;,; $19.64 ;,;

white;,;09/15/17 , Jacob Kennedy ;,; $11.40

;,; white&red ;,; 09/15/17, Craig Chambers;,;

$8.79 ;,; white&blue&red ;,;09/15/17 , Peggy Bell;,; $8.65 ;,;blue ;,; 09/15/17, Kenneth Cunningham ;,; $10.53;,; green&blue ;,;

09/15/17 , Marvin Morgan;,; $16.49;,;

green&blue&red ;,; 09/15/17 ,Marjorie Russell

;,; $6.55 ;,; green&blue&red;,; 09/15/17 ,

Israel Cummings;,; $11.86 ;,;black;,;

09/15/17, June Doyle ;,; $22.29 ;,;

black&yellow ;,;09/15/17 , Jaime Buchanan ;,;

$8.35;,; white&black&yellow ;,; 09/15/17,

Rhonda Farmer;,;$2.91 ;,; white&black&yellow

;,;09/15/17, Darren Mckenzie ;,;$22.94;,;green

;,;09/15/17,Rufus Malone;,;$4.70 ;,; green&yellow

;,; 09/15/17 ,Hubert Miles;,; $3.59

;,;green&yellow&blue;,; 09/15/17 , Joseph Bridges ;,;$5.66 ;,; green&yellow&purple&blue

;,; 09/15/17 , Sergio Murphy ;,;$17.51 ;,;

black ;,; 09/15/17 , Audrey Ferguson ;,;

$5.54;,;black&blue ;,;09/15/17 ,Edna Williams ;,;

$17.13;,; black&blue;,; 09/15/17, Randy Fleming;,; $21.13 ;,;black ;,;09/15/17 ,Elisa Hart;,; $0.35 ;,; black&purple;,; 09/15/17 ,

Ernesto Hunt ;,; $13.91 ;,; black&purple ;,;

09/15/17, Shannon Chavez ;,;$19.26 ;,;

yellow;,; 09/15/17 , Sammy Cain;,; $5.45;,;

yellow&red ;,;09/15/17 , Steven Reeves ;,;$5.50

;,; yellow;,; 09/15/17, Ruben Jones ;,;

$14.56 ;,; yellow&blue;,;09/15/17 , Essie Hansen;,; $7.33 ;,; yellow&blue&red

;,; 09/15/17 , Rene Hardy ;,; $20.22 ;,;

black ;,; 09/15/17 , Lucy Snyder ;,; $8.67

;,;black&red ;,; 09/15/17 ,Dallas Obrien ;,;

$8.31;,; black&red ;,; 09/15/17, Stacey Payne

;,; $15.70 ;,; white&black&red ;,;09/15/17

, Tanya Cox ;,; $6.74 ;,;yellow ;,;

09/15/17 , Melody Moran ;,; $30.84

;,;yellow&black;,; 09/15/17 , Louise Becker ;,;

$12.31 ;,; green&yellow&black;,; 09/15/17 ,

Ryan Webster;,;$2.94 ;,; yellow ;,; 09/15/17

,Justin Blake ;,; $22.46 ;,;white&yellow ;,;

09/15/17, Beverly Baldwin ;,; $6.60;,;

white&yellow&black ;,;09/15/17 , Dale Brady

;,; $6.27 ;,; yellow ;,;09/15/17 ,Guadalupe Potter ;,;$21.12 ;,; yellow;,; 09/15/17 ,

Desiree Butler ;,;$2.10 ;,;white;,; 09/15/17

,Sonja Barnett ;,; $14.22 ;,;white&black;,;

09/15/17, Angelica Garza;,;$11.60;,;white&black

;,; 09/15/17 , Jamie Welch ;,; $25.27 ;,;

white&black&red ;,;09/15/17 , Rex Hudson

;,;$8.26;,; purple;,; 09/15/17 , Nadine Gibbs

;,; $30.80 ;,; purple&yellow ;,; 09/15/17 ,

Hannah Pratt;,; $22.61 ;,; purple&yellow

;,;09/15/17,Gayle Richards;,;$22.19 ;,;

green&purple&yellow ;,;09/15/17 ,Stanley Holland

;,; $7.47 ;,; red ;,; 09/15/17 , Anna Dean;,;$5.49 ;,; yellow&red ;,; 09/15/17 ,

Terrance Saunders ;,; $23.70 ;,;green&yellow&red

;,; 09/15/17 , Brandi Zimmerman ;,; $26.66 ;,;

red ;,;09/15/17 ,Guadalupe Freeman ;,; $25.95;,;

green&red ;,; 09/15/17 ,Irving Patterson

;,;$19.55 ;,; green&white&red ;,; 09/15/17 ,Karl Ross;,; $15.68;,; white ;,; 09/15/17 , Brandy Cortez ;,;$23.57;,; white&red ;,;09/15/17,

Mamie Riley ;,;$29.32;,; purple;,;09/15/17 ,Mike Thornton ;,; $26.44 ;,; purple ;,; 09/15/17,

Jamie Vaughn ;,; $17.24;,;green ;,; 09/15/17 ,

Noah Day ;,; $8.49 ;,;green ;,;09/15/17

,Josephine Keller ;,;$13.10 ;,;green;,; 09/15/17 , Tracey Wolfe;,;$20.39 ;,; red ;,; 09/15/17 ,

Ignacio Parks;,;$14.70 ;,; white&red ;,;09/15/17

, Beatrice Newman ;,;$22.45 ;,;white&purple&red

;,; 09/15/17, Andre Norris ;,; $28.46 ;,;

red;,; 09/15/17 , Albert Lewis ;,; $23.89;,;

black&red;,; 09/15/17, Javier Bailey ;,;

$24.49 ;,; black&red ;,; 09/15/17 , Everett Lyons ;,;$1.81;,; black&red ;,; 09/15/17 ,

Abraham Maxwell;,; $6.81 ;,;green;,; 09/15/17

, Traci Craig ;,;$0.65;,; green&yellow;,;

09/15/17 , Jeffrey Jenkins ;,;$26.45;,;

green&yellow&blue ;,; 09/15/17, Merle Wilson

;,; $7.69 ;,; purple;,; 09/15/17,Janis Franklin

;,;$8.74 ;,; purple&black ;,;09/15/17 ,

Leonard Guerrero ;,; $1.86 ;,;yellow

;,;09/15/17,Lana Sanchez;,;$14.75 ;,; yellow;,;

09/15/17 ,Donna Ball ;,; $28.10 ;,;

yellow&blue;,; 09/15/17 , Terrell Barber ;,;

$9.91 ;,; green ;,;09/15/17 ,Jody Flores;,;

$16.34 ;,; green ;,; 09/15/17, Daryl Herrera

;,;$27.57;,; white;,; 09/15/17 , Miguel Mcguire;,;$5.25;,; white&blue ;,; 09/15/17 ,

Rogelio Gonzalez;,; $9.51;,; white&black&blue

;,; 09/15/17 , Lora Hammond ;,;$20.56 ;,;

green;,; 09/15/17,Owen Ward;,; $21.64 ;,;

green&yellow;,;09/15/17,Malcolm Morales ;,;

$24.99 ;,; green&yellow&black;,; 09/15/17 ,

Eric Mcdaniel ;,;$29.70;,; green ;,; 09/15/17

,Madeline Estrada;,; $15.52;,;green;,; 09/15/17

, Leticia Manning;,;$15.70 ;,; green&purple;,;

09/15/17 , Mario Wallace ;,; $12.36 ;,;green ;,;

09/15/17,Lewis Glover;,; $13.66 ;,;

green&white;,;09/15/17, Gail Phelps ;,;$30.52

;,; green&white&blue ;,; 09/15/17 , Myrtle Morris

;,; $22.66 ;,; green&white&blue;,;09/15/17"""

#------------------------------------------------

# Part 1: Step 1: Inspect the string above

# Start coding below!

# Part 1: Step 2:

daily\_sales\_replaced = daily\_sales.replace(';,;',';')

# Part 1: Step 3:

daily\_transactions = daily\_sales\_replaced.split(',')

# Part 1: Step 4:

# print(daily\_transactions)

# Part 1: Step 5:

daily\_transactions\_split = []

# Part 1: Step 6:

for transaction in daily\_transactions:

daily\_transactions\_split\

.append(transaction.split(';'))

# Part 1: Step 7

# print(daily\_transactions\_split)

# Part 1: Step 8

transactions\_clean = []

for transaction in daily\_transactions\_split:

transaction\_stripped = []

for item in transaction:

transaction\_stripped.append(item.strip())

transactions\_clean.append(transaction\_stripped)

# Part 1: Step 9

# print(transactions\_clean)

# Part 1: Step 10

customers = []

sales = []

thread\_sold = []

# Part 1: Step 11

for transaction in transactions\_clean:

customers.append(transaction[0])

sales.append(transaction[1])

thread\_sold.append(transaction[2])

# Part 1: Step 12

# print(customers)

# print(sales)

# print(thread\_sold)

# Part 2: Step 1

total\_sales = 0

# Part 2: Step 2

for item in sales:

total\_sales += float(item.strip('$'))

# Part 2: Step 3

print(total\_sales)

# Part 3: Step 1

# print(thread\_sold)

# Part 3: Step 2

thread\_sold\_split = []

# Part 3: Step 3

for item in thread\_sold:

if not '&' in item:

thread\_sold\_split.append(item)

else:

for i in item.split('&'):

thread\_sold\_split.append(i)

# Part 3: Step 4

def color\_count(color):

counter = 0

for item in thread\_sold\_split:

if item == color:

counter += 1

return counter

# Part 3: Step 5

print(color\_count('white'))

# Part 3: Step 6

colors = ['red','yellow','green','white','black','blue','purple']

# Part 3: Step 7

for color in colors:

print("Thread Shed sold {} threads of {} thread today.".format(color\_count(color), color))

# Strings - Code Challenge

Now you have learned about the basics of Python syntax, control flow, lists, loops, string methods and how to create functions that run repeatable blocks of code.

Let's reinforce these concepts with a series of practice problems.

# Introduction

This lesson will help you review Python functions by providing some challenge exercises involving Strings.

As a refresher, function syntax looks like this:

def some\_function(some\_input1, some\_input2):

… do something with the inputs …

return output

For example, a function that finds the difference in length between two Strings would look like this:

def lengthDiff(str1, str2):

return len(str1) - len(str2)

And this would produce output like:

>>> lengthDiff("Python", "rocks")

1

>>> lengthDiff("Marco", "Polo")

1

>>> lengthDiff("Kevin", "Durant")

-1

When you're ready to do this series of short function challenges, continue on to the rest of the lesson!

# Count Letters

unique\_english\_letters(word)

**1.**

Write a function called unique\_english\_letters that takes the string word as a parameter. The function should return the total number of unique letters in the string. Uppercase and lowercase letters should be counted as different letters.

We've given you a list of every uppercase and lower case letter in the English alphabet. It will be helpful to include that list in your function.

Hint: Loop through the list of English letters and check to see if each letter is included in word by using in.

**Solution:**

letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"

# Write your unique\_english\_letters function here:

def unique\_english\_letters(word):

uniques = 0

for letter in letters:

if letter in word:

uniques += 1

return uniques

# Uncomment these function calls to test your tip function:

print(unique\_english\_letters("mississippi"))

# should print 4

print(unique\_english\_letters("Apple"))

# should print 4

# Count X

count\_char\_x()

Instructions

**1.**

Write a function named count\_char\_x that takes a string named word and a single character named x as parameters. The function should return the number of times x appears in word.

**Hint:**

Use a for loop to loop through all of the characters of word. If the letter is equal to the value of x, increase a counter variable by one.

**Solution:**

# Write your count\_char\_x function here:

def count\_char\_x(word, x):

occurrences = 0

for letter in word:

if letter == x:

occurrences += 1

return occurrences

# Uncomment these function calls to test your tip function:

print(count\_char\_x("mississippi", "s"))

# should print 4

print(count\_char\_x("mississippi", "m"))

# should print 1

# Count Multi X

count\_multi\_char\_x()

Instructions

**1.**

Wire a function named count\_multi\_char\_x that takes a string named word and a string named x. This function should do the same thing as the count\_char\_x function you just wrote - it should return the number of times x appears in word. However, this time, make sure your function works when x is multiple characters long.

For example, count\_multi\_char\_x("Mississippi", "iss") should return 2

Hint:

Consider using the split function. How does the length of word.split(x) relate to the number of times x was in word?

**Solution:**

# Write your count\_multi\_char\_x function here:

def count\_multi\_char\_x(word, x):

splits = word.split(x)

return(len(splits)-1)

# Uncomment these function calls to test your function:

print(count\_multi\_char\_x("mississippi", "iss"))

# should print 2

print(count\_multi\_char\_x("apple", "pp"))

# should print 1

# Substring Between

substring\_between\_letters()

**1.**

Write a function named substring\_between\_letters that takes a string named word, a single character named start, and another character named end. This function should return the substring between the first occurrence of start and end in word. If start or end are not in word, the function should return word.

For example, substring\_between\_letters("apple", "p", "e") should return "pl".

**Hint:**

Begin by finding the indices of the start and end characters by using word.find(start) and word.find(end).

If either of those indices are -1, then the original string didn't contain one of those characters, and you should return word.

If neither are -1, then slice word using those indices. Remember, slicing is [inclusive:exclusive]!

**Solution:**

# Write your substring\_between\_letters function here:

def substring\_between\_letters(word, start, end):

start\_ind = word.find(start)

end\_ind = word.find(end)

if start\_ind > -1 and end\_ind > -1:

return(word[start\_ind+1:end\_ind])

return word

# Uncomment these function calls to test your tip function:

print(substring\_between\_letters("apple", "p", "e"))

# should print "pl"

print(substring\_between\_letters("apple", "p", "c"))

# should print "apple"

# X Length

x\_length\_words()

**1.**

Create a function called x\_length\_words that takes a string named sentence and an integer named x as parameters. This function should return True if every word in sentence has a length greater than or equal to x.

**Hint:**

First create a list of every word in sentence by using sentence.split(). Then iterate through that list and if any of the words have a length less than x, return False. If you iterate through all of the words and haven't returned False, you know every word had a length greater than or equal to x, so you should return True.

**Solution:**

# Write your x\_length\_words function here:

def x\_length\_words(sentence, x):

words = sentence.split(" ")

for word in words:

if len(word) < x:

return False

return True

# Uncomment these function calls to test your function:

print(x\_length\_words("i like apples", 2))

# should print False

print(x\_length\_words("he likes apples", 2))

# should print True

# Check Name

check\_for\_name()

**1.**

Write a function called check\_for\_name that takes two strings as parameters named sentence and name. The function should return True if name appears in sentence in all lower case letters, all uppercase letters, or with only the first letter capitalized. The function should return False otherwise

For example, the following three calls should all return True:

check\_for\_name("My name is Jamie", "Jamie")

check\_for\_name("My name is jamie", "Jamie")

check\_for\_name("My name is JAMIE", "Jamie")

**Hint:**

name.lower() in sentence.lower() will help you find out if the name is the sentence.

**Solution:**

# Write your check\_for\_name function here:

def check\_for\_name(sentence, name):

return name.lower() in sentence.lower()

# Uncomment these function calls to test your function:

print(check\_for\_name("My name is Jamie", "Jamie"))

# should print True

print(check\_for\_name("My name is jamie", "Jamie"))

# should print True

print(check\_for\_name("My name is Samantha", "Jamie"))

# should print False

# Every Other Letter

every\_other\_letter()

**1.**

Create a function named every\_other\_letter that takes a string named word as a parameter. The function should return a string containing every other letter in word.

**Hint:**

The following code will print all letters of a string by index:

my\_string = "Hello World"

for i in range(len(my\_string)):

print my\_string[i]

In this code, i starts at 0 and increase until it is once less than the length of my\_string. How could you make i increase by more than one each time?

Additionally, instead of printing each individual letter, you should add each letter to a new string using +.

**Solution:**

# Write your every\_other\_letter function here:

def every\_other\_letter(word):

every\_other = ""

for i in range(0, len(word), 2):

every\_other += word[i]

return every\_other

# Uncomment these function calls to test your tip function:

print(every\_other\_letter("Codecademy"))

# should print Cdcdm

print(every\_other\_letter("Hello world!"))

# should print Hlowrd

print(every\_other\_letter(""))

# should print

# Reverse

reverse\_string()

**1.**

Write a function named reverse\_string that has a string named word as a parameter. The function should return word in reverse.

**Hint:**

Just like the last challenge, you want to access each letter of word by it's index.

my\_string = "Hello World"

for i in range(len(my\_string)):

print my\_string[i]

However, you don't want i to start at 0. Instead you want it to start at the last index of your string (len(my\_string)-1) and end at 0.

Edit the call to the range function to do this. Remember, the range function can take three parameters: the starting number (inclusive), the ending number (exclusive), and the step. To count down, make the step -1.

**Solution:**

# Write your reverse\_string function here:

def reverse\_string(word):

reverse = ""

for i in range(len(word)-1, -1, -1):

reverse += word[i]

return reverse

# Uncomment these function calls to test your function:

print(reverse\_string("Codecademy"))

# should print ymedacedoC

print(reverse\_string("Hello world!"))

# should print !dlrow olleH

print(reverse\_string(""))

# should print

# Make Spoonerism

make\_spoonerism()

**1.**

A [Spoonerism](https://en.wikipedia.org/wiki/Spoonerism) is an error in speech when the first syllables of two words are switched. For example, a Spoonerism is made when someone says "Belly Jeans" instead of "Jelly Beans".

Write a function called make\_spoonerism that takes two strings as parameters named word1 and word2. Finding the first syllable of a word is a difficult task, so for our function, we're going to switch the first letters of each word. Return the two new words as a single string separated by a space.

**Hint:**

word2[0] will access the first letter of word2. word1[1:] will access everything but the first letter of word1. Combining those with a + will give you your first new word.

**Solution:**

# Write your make\_spoonerism function here:

def make\_spoonerism(word1, word2):

return word2[0]+word1[1:]+" "+word1[0]+word2[1:]

# Uncomment these function calls to test your tip function:

print(make\_spoonerism("Codecademy", "Learn"))

# should print Lodecademy Cearn

print(make\_spoonerism("Hello", "world!"))

# should print wello Horld!

print(make\_spoonerism("a", "b"))

# should print b a

# Add Exclamation

add\_exclamation()

Instructions

**1.**

Create a function named add\_exclamation that has one parameter named word. This function should add exclamation points to the end of word until word is 20 characters long. If word is already at least 20 characters long, just return word.

**Hint:**

Use a while loop to add exclamation points to word. The while loop should stop when the length of word is greater than or equal to 20.

**Solution:**

# Write your add\_exclamation function here:

def add\_exclamation(word):

while(len(word) < 20):

word += "!"

return word

# Uncomment these function calls to test your function:

print(add\_exclamation("Codecademy"))

# should print Codecademy!!!!!!!!!!

print(add\_exclamation("Codecademy is the best place to learn"))

# should print Codecademy is the best place to learn

# Coded Communication

About 2 minutes

In this off-platform project, you will be exchanging coded communications with your pen pal, Vishal. The two of you challenge each other by sending coded messages that you will have to use your Python skills to decipher! Put your programming skills to the test with these fun cryptography puzzles.

**This project is totally optional; if you’re busy this week, you can skip it!**

Follow the steps below to get started with your project!

#### Working on Your Computer

1. If you’ve never used the command line, we recommend taking the [Learn the Command Line course](https://www.codecademy.com/learn/learn-the-command-line).
2. Install Python by following the directions in this article on [Installing Python](https://www.codecademy.com/articles/install-python3).
3. Learn about [Jupyter Notebooks](https://www.codecademy.com/articles/how-to-use-jupyter-notebooks-py3), a cool way of combining Python code with explanations or instruction in a web terminal.
4. Download the project: [Coded Correspondence](https://s3.amazonaws.com/codecademy-content/programs/programming-with-python/coded_correspondence.zip)
5. Unzip it by double-clicking on it.
6. In the terminal, navigate to the directory containing the project, and type:

jupyter notebook

This should open a browser tab.

1. Click on coded\_correspondence.ipynb in the browser tab. This will open up your Jupyter Notebook.
2. Follow the steps in the Jupyter Notebook. If you get stuck, you can look at coded\_correspondence\_solutions.ipynb for the answer.

**This is a challenging project! Be prepared to come up with your own creative methods of solving the problems!**

**See file coded\_coorespondence using Jupyter UI.**