**INTRODUCTION TO STRINGS**

**Introduction to Strings**

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**](https://www.codecademy.com/resources/docs/python/strings?page_ref=catalog). A string is a sequence of characters contained within a pair of 'single quotes' or "double quotes". A string can be any length and can contain any letters, numbers, symbols, and spaces.

In this lesson, we will learn more about strings and how they are treated in Python. We 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.

**Instructions**

**1.**

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

Checkpoint 2 Passed

Hint

Strings are written inside either single quotes ' ' or double quotes " ". Just be sure they match!

# Valid string using double quotes  
favorite\_word = "coding"  
   
# Also valid string using single quotes  
favorite\_word = 'coding'

**2.**

Print favorite\_word.

Checkpoint 3 Passed

Hint

It should look something like:

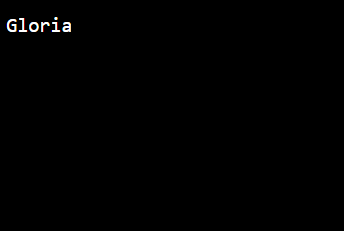
print(string\_name)

Notice how the quotes don’t appear in the output

**script.py**

favorite\_word = "Gloria"

print(favorite\_word)

****

**They're all Lists!**

A string can be thought of as a [**list**](https://www.codecademy.com/resources/docs/python/lists?page_ref=catalog) 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.

print(favorite\_fruit[1])  
# Output: 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:

print(favorite\_fruit[0])  
# Output: b

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

For example:

print(favorite\_fruit[1.5])

This would result in:

Traceback (most recent call last):

File "script.py", line 3, in <module>

print(favorite\_fruit[1.5])

TypeError: string indices must be integers

**Instructions**

**1.**

One of the most common things that are represented by strings is names.

Save your name as a string to the variable my\_name.

Checkpoint 2 Passed

Hint

Remember, the string can include more than one word:

my\_name = "Python Person"

**2.**

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

Checkpoint 3 Passed

Hint

The first character of a string is at index 0:

first\_initial = my\_name[0]

**script.py**

my\_name = "Andres"

first\_initial = my\_name[0]

**Cut Me a Slice of String**

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

string[first\_index:last\_index]

This is called *slicing* a string. When we slice a string we are creating a [substring](https://www.codecademy.com/resources/docs/python/substrings?page_ref=catalog) - a brand 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 be. We could slice favorite\_fruit as follows:

print(favorite\_fruit[4:6])  
# Output: be

Notice how the character at the first index, b, is included, but the character at the last index, r, is excluded. If you look for the indices 4 and 6 in the diagram, you can see how the r 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.

print(favorite\_fruit[:4])  
# Output: blue  
   
print (favorite\_fruit[4:])  
# Output: berry

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

**Instructions**

**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.

Checkpoint 2 Passed

Hint

It should look something like:

new\_account = last\_name[:5]

**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 last\_name. Your string should have a total of 4 characters.

Checkpoint 3 Passed

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]

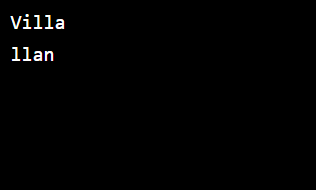
**script.py**

irst\_name = "Rodrigo"

last\_name = "Villanueva"

new\_account = last\_name[:5]

temp\_password = last\_name[2:6]

****

**Concatenating Strings**

We can also *concatenate*, or combine, 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  
   
print(favorite\_fruit)  
# Output: blueberries

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

fruit\_sentence = "My favorite fruit is" + favorite\_fruit  
   
print(fruit\_sentence)  
# Output: My favorite fruit isblueberries  
   
fruit\_sentence = "My favorite fruit is " + favorite\_fruit  
   
print(fruit\_sentence)  
# Output: My favorite fruit is blueberries

It’s subtle, but notice that in the first example, there is no space between “is” and “blueberries”.

**Instructions**

**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.

Checkpoint 2 Passed

Hint

Remember, a new function can be defined using:

def function\_name(parameter):  
  # Code goes here

**2.**

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

**script.py**

first\_name = "Julie"

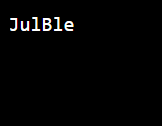
last\_name = "Blevins"

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

  return first\_name[:3] +  last\_name[:3]

new\_account = account\_generator("Julie", "Blevins")

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"  
   
length = len(favorite\_fruit)  
   
print(length)  
# Output: 9

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

fruit\_sentence = "I love blueberries"  
   
print(len(fruit\_sentence))  
# Output: 18

len() comes in handy when we are trying to select characters from the end of a string. What is the index of the last character,"y", of favorite\_fruit from above? You can try to run the following code:

last\_char = favorite\_fruit[len(favorite\_fruit)]  
   
print(last\_char)

This will result in:

IndexError: string index out of range

Why does this generate an IndexError? Because the indices start at 0, so the final character in favorite\_fruit has an index of 8. len(favorite\_fruit) returns 9 and, because there is no value at that index, an IndexError occurs.

Instead, the last character in a string has an index that is len(string\_name) - 1.

last\_char = favorite\_fruit[len(favorite\_fruit)-1]  
   
print(last\_char)  
# Output: y

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

length = len(favorite\_fruit)  
last\_chars = favorite\_fruit[length-4:]  
print(last\_chars)  
# Output: 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().

**Instructions**

**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 concatenates the last three letters of each and returns them as a string.

Checkpoint 2 Passed

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.

**script.py**

first\_name = "Reiko"

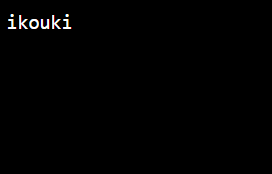
last\_name = "Matsuki"

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

  return first\_name[(len(first\_name) - 3):] + last\_name[(len(last\_name) - 3):]

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

print(temp\_password)

****

**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'  
print(favorite\_fruit[-1])  
# => 'y'  
   
print(favorite\_fruit[-2])  
# => 'r'  
   
print(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 second to last character in company\_motto. Save this to the variable second\_to\_last.

Checkpoint 2 Passed

**2.**

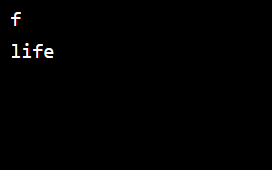
Use negative indices to create a slice of the last 4 characters in company\_motto. Save this to the variable final\_word.

**script.py**

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.

**Instructions**

**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"

Checkpoint 2 Passed

**2.**

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

Delete the code you just wrote for step 1.

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

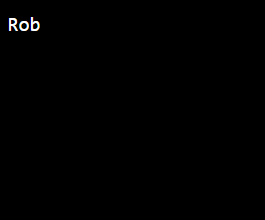
**script.py**

first\_name = "Bob"

last\_name = "Daily"

fixed\_first\_name = "R" + first\_name[1:]

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!

**Instructions**

**1.**

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

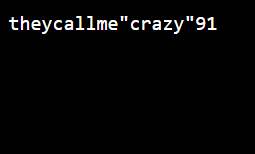
theycallme"crazy"91

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.

**script.py**

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 of problems where we need to iterate through a string.

**Instructions**

**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()!

Checkpoint 2 Passed

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

**script.py**

def get\_length(string):

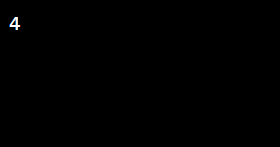
  counter = 0;

  for letter in string:

    counter += 1

  return counter

print(get\_length("hola"))

****

**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.

**Instructions**

**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.

Checkpoint 2 Passed

Hint

Make sure to print() the letter\_check() function when testing it. To test try running:

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

and

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

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

**script.py**

def letter\_check(word, letter):

  checker = False

  for litera in word:

    if litera == letter:

      checker = True

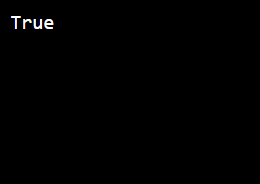
      break

    else:

      checker = False

  return checker

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

****

**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:

print("e" in "blueberry")  
# => True  
print("a" in "blueberry")  
# => False

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

print("blue" in "blueberry")  
# => True  
print("blue" in "strawberry")  
# => False

**Instructions**

**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.

Checkpoint 2 Passed

**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'].

Checkpoint 3 Passed

Hint

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

Also, make sure to test your function!

**script.py**

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

  if little\_string in big\_string:

    return True

  else:

    return False

print(contains("watermelon", "melon"))

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

  modified\_string\_two = []

  subresult = []

  for letter in string\_two:

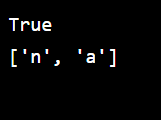
    if letter in string\_one:

      subresult.append(letter)

  result = set(subresult)

  return list(result)

print(common\_letters('manhattan', 'san francisco'))

****