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**. 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`.

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`.

Hint

It should look something like:

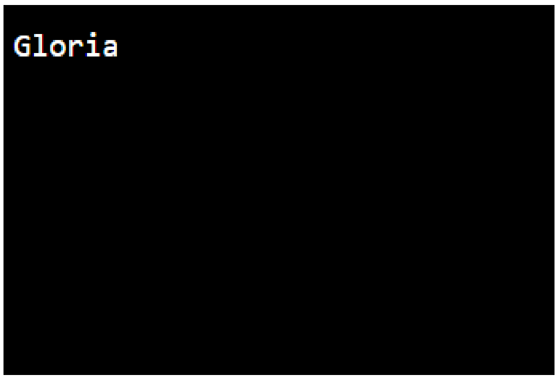
```
print(string_name)
```

Notice how the quotes don't appear in the output

script.py

```
favorite_word = "Gloria"

print(favorite_word)
```



```
Gloria
```

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.

```
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`.

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`.

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 - 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`.

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.

Hint

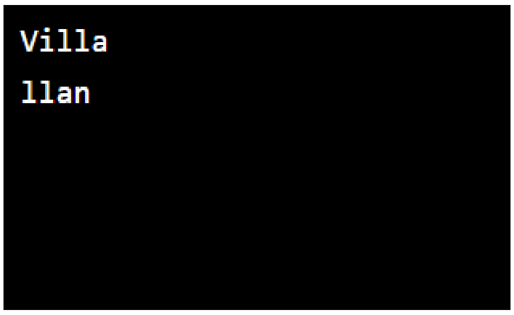
Remember, because indices start at 0, to get the third through sixth characters of a string you would want to use

```
string_name[2:6]
```

script.py

```python
irst_name = "Rodrigo"
last_name = "Villanueva"

new_account = last_name[:5]
temp_password = last_name[2:6]
```



```
Villa
llan
```

Concatenating Strings

We can also *concatenate*, or combine, two existing strings together into a new string. Consider the following two strings:

```python
fruit_prefix = "blue"
fruit_suffix = "berries"
```

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

```python
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.

```python
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.

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`.

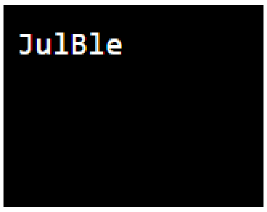
script.py

```
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)
```



```
JulBle
```

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.

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`.

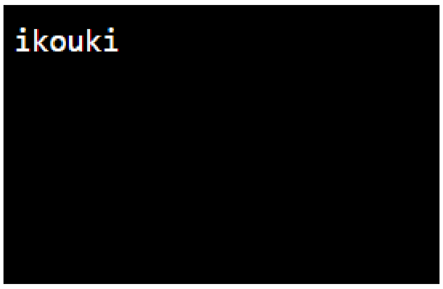
script.py

```
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)
```



```
ikouki
```

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:

```python
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

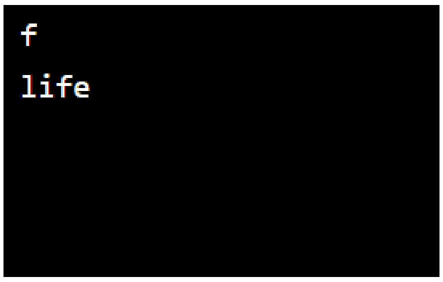
1.
Use negative indices to find the second to last character in `company_motto`. Save this to the variable `second_to_last`.

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

```python
company_motto = "Copeland's Corporate Company helps you capably cope with the con
stant cacophony of daily life"

second_to_last = company_motto[-2]
final_word = company_motto[-4:]
```



```
f
life
```

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"
```

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.

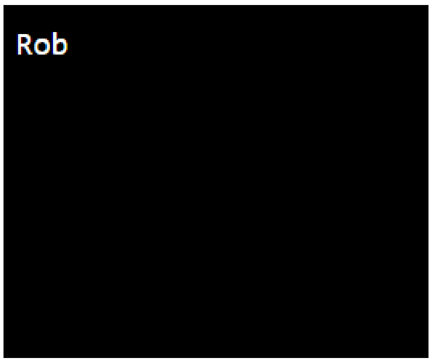
Then, concatenate the string `"R"` with a slice of `first_name` that includes everything but the first character, `"B"`, and save it to a new string `fixed_first_name`.

script.py

```
first_name = "Bob"
last_name = "Daily"

fixed_first_name = "R" + first_name[1:]

print(fixed_first_name)
```


```
Rob
```

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.

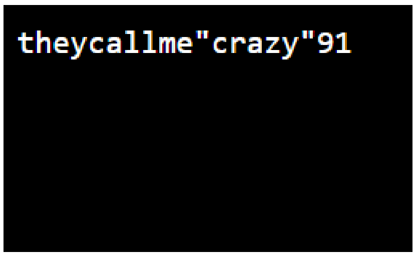
When Rob Daily (remember him? From the last exercise?) set up his account he set his password to be

```
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)
```



```
theycallme"crazy"91
```

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()`!

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"))
```

```
4
```

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 `b`s 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 `b`s 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.

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"))
```

```
True
```

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`.

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!

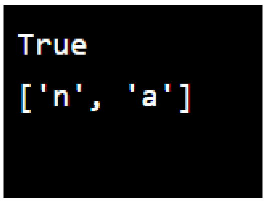
script.py

```python
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'))
```



```
True
['n', 'a']
```

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!

Instructions

1.
Copeland’s Corporate Company has finalized what they want 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 `user_name`. 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 a function called `password_generator` that takes one parameter `user_name` and defines an empty string named `password` within the function body.

3.
Inside `password_generator`, create a `for` loop that iterates through the indices of `user_name` by going from `0` to `len(user_name)`.

The loop should create the `password` by shifting all the letters of `user_name` one to the right. To do so, add the letter at the *previous* index of `user_name` to `password` with each pass of the loop.

After the `for` loop but still within the definition of `password_generator`, return the `password`.

Hint

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

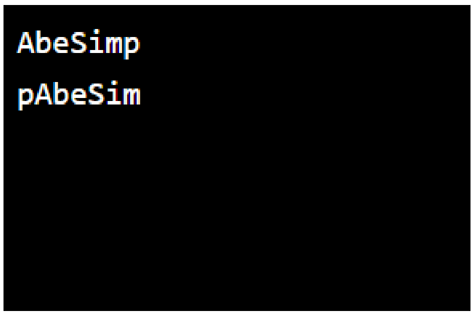
For example, after the `for` loop is complete, `print(password_generator("apple"))` should print `eappl`.

script.py

```python
def username_generator(first_name, last_name):
  if len(first_name) < 3 or len(last_name) < 4:
    user_name = first_name + last_name
  else:
    user_name = first_name[:3] + last_name[:4]
  return user_name

def password_generator(user_name):
  password = user_name[-1] + user_name[0:len(user_name) -1]
  return password

print(username_generator("Abe", "Simpson"))
username = username_generator("Abe", "Simpson")
print(password_generator(username))
```



```
AbeSimp
pAbeSim
```