Chapter 6 - Strings

Strings

String Basics

Learning Objectives - String Basics

- **Identify the three properties of strings**
- **Recognize that strings are immutable**
- **Write a multiline string**
- **Calculate the result from slicing a string**
- **Utilize escape characters to add special characters to a string**

String Properties

String Length

We have already seen strings in the “Fundamentals”section. We are going to dig a little deeper with this data type. All strings have the following characteristics:

1. **Characters** - Strings are made up of characters between quotation marks (previously covered in the “Fundamentals” section).
2. **Length** - Each string has a length (total number of characters).
3. **Index** - Each character in a string has a position, called an index.

To calculate the length of a string, use the `len` function. This function will return an integer that is the sum of all of the characters between the quotation marks.

```
my_string = "Hello"
length = len(my_string)
print(length)
```

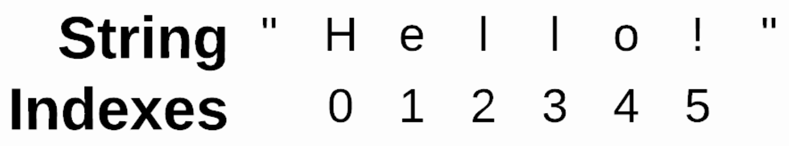
challenge

What happens if you:

- Change `my_string` to `"Hello world!"`?
- Change `my_string` to `""`?
- Change `my_string` to `"-1"`?

String Index

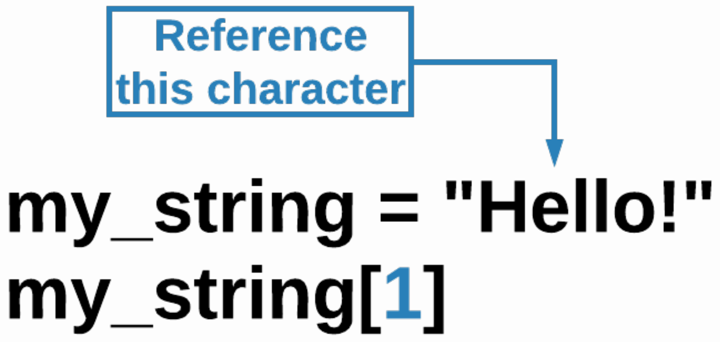
Each character in a string has a position. This is its index. Indexes always start with 0.



String Index

▶ **Strings & Quotation Marks**

To reference a character, use the string name, followed by square brackets (`[]`), and put the index between the square brackets.



Referencing a Character with an Index

```
my_string = "Hello!"
character = my_string[1]
print(character)
```

challenge

What happens if you:

- Change `character` to `my_string[len(my_string)]`?
- Change `character` to `my_string[len(my_string) - 1]`?
- Change `character` to `my_string[-1]`?

Immutability

Immutability

You now know how to reference each character of a string. What do you think the code below will do?

```
my_string = "House"
my_string[0] = "M"
print(my_string)
```

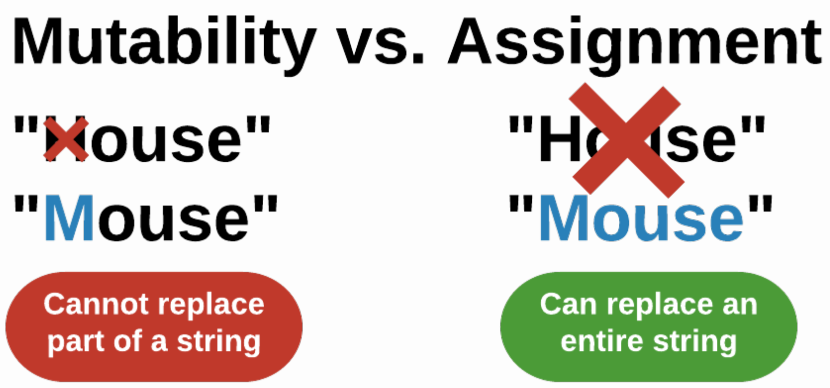
If you thought the code above would print `Mouse`, that would be a logical guess. However, you see an error. Strings are immutable. That means you cannot change their value.

Yes, but…

The code below works just fine. Isn’t this an example of changing the value of a string?

```
my_string = "House"
my_string = "Mouse"
print(my_string)
```

Python is doing something very subtle behind the scenes. The first example on this page is about mutability. That is, changing just a part of a whole. The second example is about the assignment operator. Assignment replaces the entire value with a new value. So, you can replace an entire string (assignment), but you cannot change part of a string (mutability). That is why strings are considered to be immutable.


Mutability vs. Assignment
"House"
"Mouse"
Cannot replace part of a string
"House"
"Mouse"
Can replace an entire string

Mutability vs Assignment

Multiline Strings

Multiline Strings

Python has several informal rules. Breaking one of these will not cause an error, but the Python community will not consider the code to be “proper”. One of these rules is that a line of text should not have more than 79 characters. If a string has more than 79 characters, use the newline character (\) and go to the next line. Note, Python will print the string as one line of text.

```
my_string = "Hello world! This is a very, very long string. \
Even though this string is on three different lines, it should \
print as one line. Notice how the line breaks are different."
print(my_string)
```

challenge

What happens if you:

- Put a space after the ``?

Triple Quotes

Triple quotation marks can be used to preserve the whitespace of a string.

```
long_string = """Notice how this weird looking
    string is being
        printed."""
print(long_string)
```

▶ Docstrings

challenge

What happens if you:

- Change the `"""` (3 double quotes) to `'''` (3 single quotes)?
- Have `"""` (3 double quotes) to start the string and `'''` (3 single quotes) to end the string?

In Operator

The In Operator

The `in` operator tells you if a character or a string is present in another string. `in` returns a boolean value, either `True` or `False`.

```
my_string = "The brown dog jumps over the lazy fox."

print("dog" in my_string)
```

challenge

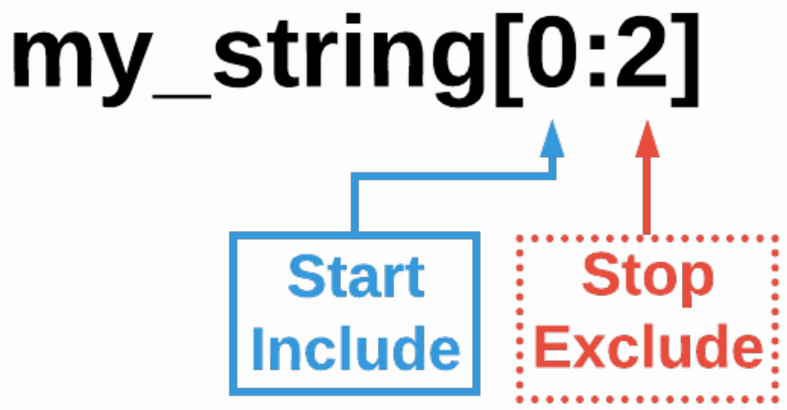
What happens if you:

- Change the `print` statement to be `print("cat" in my_string)`?
- Change the `print` statement to be `print("Dog" in my_string)`?
- Change the `print` statement to be `print(" " in my_string)`?
- Change the `print` statement to be `print(my_string in my_string)`?

The Slice Operator

The Slice Operator

The slice operator (:) returns a portion of the string. Provide numbers to the slice operator to indicate where you start and stop. The slice operator includes the first number, but does **not include** the second number. The slice operator does not modify the original string. Instead, it returns a partial copy of the original string.



String Slice

```
my_string = "The brown dog jumps over the lazy fox."
my_slice = my_string[4:9]

print(my_slice)
```

challenge

What happens if you:

- Change the slice to be `my_string[1:2]`?
- Change the slice to be `my_string[0:len(my_string)]`?
- Change the slice to be `my_string[1:1]`?
- Change the slice to be `my_string[:2]`?

Escape Characters

Escape Characters

An escape character is a character that has a different interpretation that what you see in a string. Escape characters always start with a backslash (\). The most common escape character is the newline character (\n) which causes Python to print on the next line.

```
my_string = "Hello\nworld"
print(my_string)
```

Escape Character	Description	Example
\\	Prints a backslash	print("\\")
\'	Prints a single quote	print("\'")
\"	Prints a double quote	print("\"")
**\	Prints a tab (spacing)	print("Hello\tworld")
***	Prints a hexidecimal unicode character	print("\u26BE")

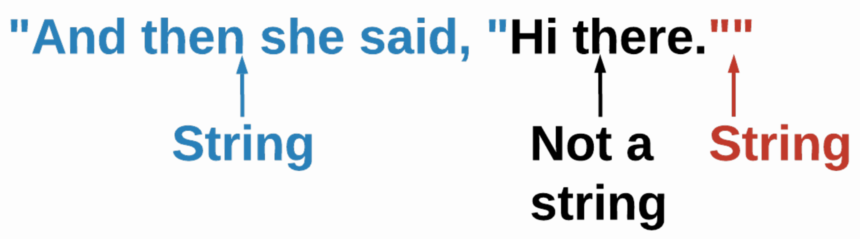
challenge

What happens if you:

- Use nn instead of n?
- Replace the nn with t?
- Find a hexidecimal unicode character to use (see link above)?

Quotes Inside Quotes

Imagine that you have this small bit of dialog, `And then she said, "Hi there."` and want to store it as a string. Typing `"And then she said, "Hi there.""` would cause an error.



Quote in a Quote Wrong

When you use a " to start a string, Python looks for the next " to end it. To avoid syntax errors, you can use a double quote to start your string, single quotes for the inner quote, and end the string with a double quote.

```
my_string = "And then she said, 'Hi there.'"
print(my_string)
```

challenge

What happens if you:

- Use single quotes (') on the outside and double quotes (") on the inside?
- Use only single quotes (')?
- Use the escape character " for the inner quotation marks?