Chapter 6 - Strings

Strings

Formatting Strings

Learning Objectives - Formatting Strings

- **Define string interpolation**
- **Identify basic forms of string interpolation**
- **Demonstrate ability to format strings with f-strings and `.format`**
- **Recognize string formatting with %**

String Interpolation

String Interpolation

String interpolation is the process of putting the value of a variable inside of a string. You have already practiced a form of string interpolation by using `+` and type casting.

```
arms = 2
fingers = 10
print("I have " + str(arms) + " arms and " + str(fingers) + " fingers.
```

Comma-Separated String Interpolation

Another way to do string interpolation is to use commas instead of the `+` operator.

```
verb = "jumps"
adjective = "lazy"
print("The brown dog ", verb, " over the ", adjective, " fox.")
```

challenge

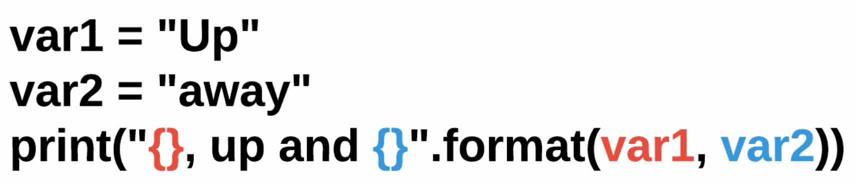
What happens if you:

- Change `verb` to `5`?
- Use commas for the variable `verb` and the `+` operator for `adjective`?
- Change `adjective` to `True`?

Format

The Format Method

All of the quotation marks, commas, and `+` can get hard to read. So Python 3 introduced a new way to perform string interpolation. The `format` method lets you put a placeholder and then assign a variable to the placeholder.



```
var1 = "Up"
var2 = "away"
print("{}, up and {}".format(var1, var2))
```

Format Method

The `{}` are placeholders for a variable, and after `format` there is a list of variables for the placeholders. Notice that the first variable goes into the first placeholder, the second variable into the second placeholder, etc.

```
var1 = "today"
var2 = "luckiest"
print("Yet {} I consider myself the {} man on the face of this earth."
```

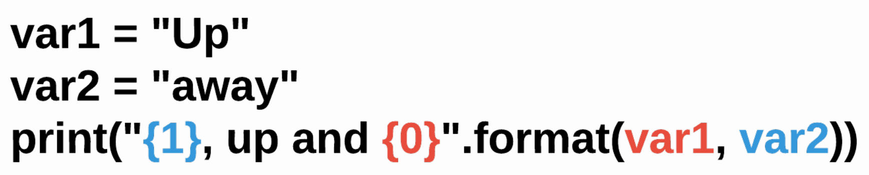
challenge

What happens if you:

- Change `format` to read `format(var2, var1)`?
- Change `var1` to `5`?
- Change `var2` to `True`?

Reorder the Variables

By default, the `format` method uses the order of variables (from left to right) when placing them into a placeholder. However, you can change the order variables by using an index. The first variable has an index of 0, the second and index of 1, etc.



```
var1 = "Up"
var2 = "away"
print("{1}, up and {0}".format(var1, var2))
```

Rearrange Order

```
var1 = "today"
var2 = "luckiest"
print("Yet {1} I consider myself the {0} man on the face of this earth
```

challenge

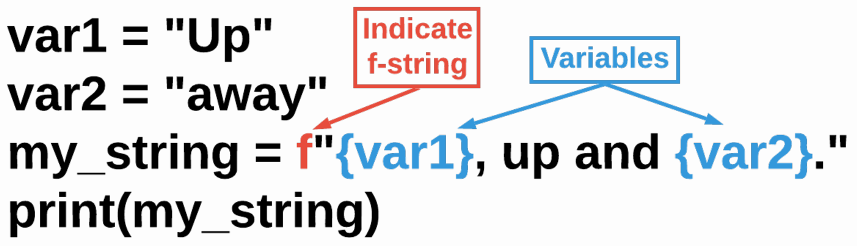
What happens if you:

- Switch the `1` and the `0`?
- Change `1` to `2`?

F-String

F-String

F-strings are a feature of Python versions 3.6 and above. F-strings are an evolution of the `format` method and simplify the process of string interpolation. F-strings start with the letter `f` followed by the string. Use `{}` to insert a variable, but put the variable between the `{}`.



F-Strings

▼ **What does the “f” mean?**

The “f” stands for fast. F-strings are executed faster than any other form of string interpolation. Here is an article about f-strings that shows the time differences for various forms of string interpolation.

```
var1 = "Up"
var2 = "away"
my_string = f"{var1}, up and {var2}."
print(my_string)
```

challenge

What happens if you:

- Change the value of `var1` to 7?
- Change the first `{}` to `{var1 + 10}`?
- Change the second `{}` to `{var2.upper()}`?

Multi-line F-Strings

You can use the same ideas from the lesson on multi-line strings to make multi-line f-strings.

```
name = "Calvin"
age = 6
occupation = "student"
sentence = f"My name is {name}. " \
           f"I am {age} years old. " \
           f"I am a {occupation}."
print(sentence)
```

challenge

What happens if you:

Use the `"""` method for multi-line strings?

```
name = "Calvin"
age = 6
occupation = "student"
sentence = f"""My name is {name}.
I am {age} years old.
I am a {occupation}."""
print(sentence)
```

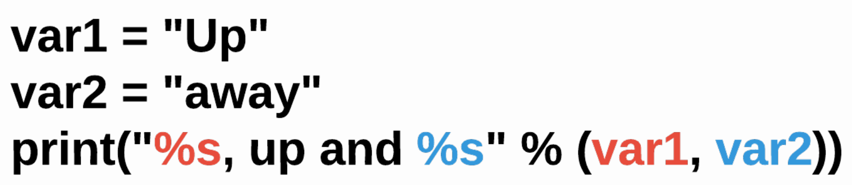
▼ **Use the appropriate number of `f`'s**

The `</code> method for making multi-line strings requires that you put an f before each set of quotation marks. The """“ method for making multi-line strings requires only one f before the ”””. That is because there is only one set of ”"". Be sure that you have an f for each set of quotation marks when making multi-line strings.`

Formatting with %

Formatting Strings with %

Important: Formatting strings with `%` is **not** the recommended way for string interpolation in Python 3. The `%` operator was commonly used for string interpolation, and has a similar syntax as the `format` method for string interpolation.



```
var1 = "Up"
var2 = "away"
print("%s, up and %s" % (var1, var2))
```

Old String Interpolation

```
var1 = "Up"
var2 = "away"
print("%s, up and %s" % (var1, var2))
```

▼
What does the `s` in `%s` mean?

The `s` means that the variable that will go in this position will be a string. If `var1` is an integer, use `%i`. If it is a float, use `%f`.

challenge

What happens if you:

- Swap the position of `var1` and `var2`?
- Remove the `s` from the `%`?
- Change `var1` to `7` and change the first `%s` to `%i`?

Why Learn the Old Way?

Python was first released in 1991. Python 3 was not released until 2008. Even then, many developers were slow to adopt Python 3. Because of this, so many examples of Python that you will see in books, websites, and videos will use `%` for string interpolation. You should be able to read and understand this code. However, you should write code using modern methods of string interpolation.