

Strings

- ✔ **Video:** Basic Structures Introduction
1 min
- ✔ **Video:** What is a string?
2 min
- ✔ **Video:** The Parts of a String
4 min
- ✔ **Reading:** String Indexing and Slicing
10 min
- ✔ **Video:** Creating New Strings
6 min
- ✔ **Reading:** Basic String Methods
10 min
- ✔ **Video:** More String Methods
4 min
- ✔ **Reading:** Advanced String Methods
10 min
- ✔ **Video:** Formatting Strings
5 min
- ✔ **Reading:** String Formatting
10 min
- ✔ **Reading:** String Reference Guide
10 min

🎧 **Reading:** Formatting Strings Guide
10 min

📖 **Reading:** Study Guide: Strings
10 min

📖 **Practice Quiz:** Practice Quiz: Strings
5 questions

Lists

Dictionaries

Module Review

Formatting Strings Guide

Python offers different ways to format strings. In the video, we explained the `format()` method. In this reading, we'll highlight three different ways of formatting strings. For this course you only need to know the `format()` method. But on the internet, you might find any of the three, so it's a good idea to know that the others exist.

Using the `format()` method

The `format` method returns a copy of the string where the `{}` placeholders have been replaced with the values of the variables. These variables are converted to strings if they weren't strings already. Empty placeholders are replaced by the variables passed to `format` in the same order.

```
1 # "base string with {} placeholders".format(variables)
2
3 example = "format() method"
4
5 formatted_string = "this is an example of using the {} on a string".format(example)
6
7 print(formatted_string)
8
9 """Outputs:
10 this is an example of using the format() method on a string
11 """
```

If the placeholders indicate a number, they're replaced by the variable corresponding to that order (starting at zero).

```
1 # "{} {} {}".format(first, second)
2
3 first = "apple"
4 second = "banana"
5 third = "carrot"
6
7 formatted_string = "{} {} {}".format(first, second, third)
8
9 print(formatted_string)
10
11 """Outputs:
12 apple carrot banana
13 """
```

If the placeholders indicate a field name, they're replaced by the variable corresponding to that field name. This means that parameters to `format` need to be passed indicating the field name.

```
1 # "{var1} {var2}".format(var1=value1, var2=value2)
```

```
1 "{:exp1} {:exp2}".format(value1, value2)
```

If the placeholders include a colon, what comes after the colon is a formatting expression. See below for the expression reference.

Official documentation for [the format string syntax](#)

```
1 # {:d} integer value
2 '{:d}'.format(10.5) → '10'
```

Formatting expressions

Expr	Meaning	Example
<code>{:d}</code>	integer value	<code>'{:d}'.format(10.5) → '10'</code>
<code>{:.2f}</code>	floating point with that many decimals	<code>'{:2f}'.format(0.5) → '0.50'</code>
<code>{:2s}</code>	string with that many characters	<code>'{:2s}'.format('Python') → 'Py'</code>
<code>{<6s}</code>	string aligned to the left that many spaces	<code>'{<6s}'.format('Py') → 'Py '</code>
<code>{>6s}</code>	string aligned to the right that many spaces	<code>'{>6s}'.format('Py') → ' Py'</code>
<code>{^6s}</code>	string centered in that many spaces	<code>'{^6s}'.format('Py') → ' Py '</code>

Check out the official documentation for [all available expressions](#).

Old string formatting (Optional)

The `format()` method was introduced in Python 2.6. Before that, the `%` (modulo) operator could be used to get a similar result. While this method is **no longer recommended** for new code, you might come across it in someone else's code. This is what it looks like:

"base string with `%s` placeholder" % variable

The `%` (modulo) operator returns a copy of the string where the placeholders indicated by `%` followed by a formatting expression are replaced by the variables after the operator.

"base string with `%d` and `%d` placeholders" % (value1, value2)

To replace more than one value, the values need to be written between parentheses. The formatting expression needs to match the value type.

"%(var1) %(var2)" % (var1:value1, var2:value2)

Variables can be replaced by name using a dictionary syntax (we'll learn about dictionaries in an upcoming video).

"Item: %s - Amount: %d - Price: %.2f" % (item, amount, price)

The formatting expressions are mostly the same as those of the `format()` method.

Check out the official documentation for [old string formatting](#).

Formatted string literals (Optional)

This feature was added in Python 3.6 and isn't used a lot yet. Again, it's included here in case you run into it in the future, but it's not needed for this or any upcoming courses.

A formatted string literal or f-string is a string that starts with `f` or `F` before the quotes. These strings might contain `{}` placeholders using expressions like the ones used for `format` method strings.

The important difference with the `format` method is that it takes the value of the variables from the current context, instead of taking the values from parameters.

Examples:

```
>>> name = "Micah"
>>> print(f"Hello {name}")

Hello Micah
```

```
>>> item = "Purple Cup"
>>> amount = 5
>>> price = amount * 3.25
>>> print(f"Item: {item} - Amount: {amount} - Price: {price:.2f}")

Item: Purple Cup - Amount: 5 - Price: 16.25
```

Check out the official documentation for [f-strings](#).

Mark as completed