Formatting Strings Cheat Sheet

Formatting Strings Cheat Sheet

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.

```
# "base string with {} placeholders".format(variables)

example = "format() method"

formatted_string = "this is an example of using the {} on a string".

print(formatted_string)

"""Outputs:
this is an example of using the format() method on a string
"""
```

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

```
# "{0} {1}".format(first, second)

first = "apple"
second = "banana"
third = "carrot"

formatted_string = "{0} {2} {1}".format(first, second, third)

print(formatted_string)

"""Outputs:
apple carrot banana
"""
```

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.

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

```
"{: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

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

Formatting expressions

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

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.