String formatting

February 6, 2021

1 String formatting

In many of the notebooks in this series of lessons, you'll see something like this:

```
name = 'Cody'
print(f'Hello, {name}!')
# Hello, Cody!
```

This is called "string formatting," and it's using a tool called "f-strings," which are available in Python 3.6 and later distributions. It's one way to pass variables to a template string. Note the `f` prepended to the string and the curly brackets `{}` placeholder with the name of the variable you'd like to inject into the string.

Here's another example:

[1]:
```
my_name = 'Cody'
my_age = 35
my_state = 'Colorado'
```

[2]:
```
greeting = f'Hello, my name is {my_name}. I am {my_age} years old, and I live␣
 ↪in {my_state}.'
```

[3]:
```
print(greeting)
```

```
Hello, my name is Cody. I am 35 years old, and I live in Colorado.
```

Another way to do the same thing is to use the `.format()` string method, which is a little more verbose:

[4]:
```
greeting_2 = 'Hello, my name is {}. I am {} years old, and I live in {}.'

print(greeting_2.format(my_name, my_age, my_state))
```

```
Hello, my name is Cody. I am 35 years old, and I live in Colorado.
```

Using f-strings is cleaner, generally, but `format()` can be more versatile in some situations because you can create a template string *before* the variable exists.

Here's an example of what I mean. Using the `format()` method:

[5]:
```
file_template = '{year}-data.csv'
```

[6]:
```
for y in range(1990, 2000):
    print(file_template.format(year=y))
```

```
1990-data.csv
1991-data.csv
1992-data.csv
1993-data.csv
1994-data.csv
1995-data.csv
1996-data.csv
1997-data.csv
1998-data.csv
1999-data.csv
```

That approach wouldn't work with an f-sring:

[7]:
```
file_template = f'{year}-data.csv'
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-215523fc40f8> in <module>
----> 1 file_template = f'{year}-data.csv'

NameError: name 'year' is not defined
```

1.0.1 Formatting numbers

Just like in Excel, you can change the formatting of a piece of data for display purposes without changing the underlying data itself. Here are a couple of the more common recipes for formatting numbers:

[8]:
```
my_number = 1902323820.823
```

Add thousand-separator commas

[9]:
```
f'{my_number:,}'
```

[9]:
```
'1,902,323,820.823'
```

Increase or decrease decimal precision

[10]:
```
# no decimal places
f'{my_number:.0f}'
```

[10]:
```
'1902323821'
```

[11]:
```
# two decimal places
f'{my_number:.2f}'
```

[11]: '1902323820.82'

[12]:
```
# two decimal places ~and~ commas
f'{my_number:,.2f}'
```

[12]: '1,902,323,820.82'

[13]:
```
# add a dollar sign to that - note that it's OUTSIDE of the curly brackets
f'${my_number:,.2f}'
```

[13]: '$1,902,323,820.82'

[14]:
```
# add a british pound sign to that
f' {my_number:,.2f}'
```

[14]: ' 1,902,323,820.82'

[15]:
```
# add an emoji to that
f' {my_number:,.2f}'
```

[15]: ' 1,902,323,820.82'

[16]:
```
# add an emoji to that ... in a sentence
f'I have  {my_number:,.2f} in GrimaceCoin, my new cryptocurrency.'
```

[16]: 'I have 1,902,323,820.82 in GrimaceCoin, my new cryptocurrency.'