Python (How to Use Str)

How To Use str.format()

str.format() is an improvement on %-formatting. It uses normal function call syntax and is extensible through the __format__() method on the object being converted to a string.

With str.format(), the replacement fields are marked by curly braces:

```
>>>

>>>"Hello, {}. You are {}.".format(name,age)
'Hello, Eric. You are 74.'
```

You can reference variables in any order by referencing their index:

```
>>>

>>>"Hello, {1}. You are {0}.".format(age,name)
'Hello, Eric. You are 74.'
```

But if you insert the variable names, you get the added perk of being able to pass objects and then reference parameters and methods in between the braces:

```
>>>

>>>person={'name':'Eric','age':74}
>>>"Hello, {name}. You are {age}.".format(name=person['name'],age=person['age'])
'Hello, Eric. You are 74.'
```

You can also use ** to do this neat trick with dictionaries:

```
>>>

>>>person={'name':'Eric','age':74}
>>>"Hello, {name}. You are {age}.".format(**person)
'Hello, Eric. You are 74.'
```

str.format() is definitely an upgrade when compared with %-formatting, but it's not all roses and sunshine.

Why str.format() Isn't Great

Code using str.format() is much more easily readable than code using %-formatting, but str.format() can still be quite verbose when you are dealing with multiple parameters and longer strings. Take a look at this:

```
>>>
```

```
>>>first_name="Eric"
>>>last_name="Idle"
>>>age=74
>>>profession="comedian"
>>>affiliation="Monty Python"
>>>print(("Hello, {first_name}{last_name}. You are {age}. "+
>>>"You are a {profession}. You were a member of {affiliation}.") \
>>>.format(first_name=first_name,last_name=last_name,age=age, \
>>>profession=profession,affiliation=affiliation))
```

'Hello, Eric Idle. You are 74. You are a comedian. You were a member of Monty Python.'
If you had the variables you wanted to pass to .format() in a dictionary, then you could just unpack it with .format(**some_dict) and reference the values by key in the string, but there has got to be a better way to do this.

f-Strings: A New and Improved Way to Format Strings in Python

The good news is that f-strings are here to save the day. They slice! They dice! They make julienne fries! Okay, they do none of those things, but they do make formatting easier. They joined the party in Python 3.6.

Also called "formatted string literals," f-strings are string literals that have an f at the beginning and curly braces containing expressions that will be replaced with their values. The expressions are evaluated at runtime and then formatted using the __format__ protocol. As always, the Python docs are your friend when you want to learn more.

Here are some of the ways f-strings can make your life easier.

Simple Syntax

The syntax is similar to the one you used with str.format() but less verbose. Look at how easily readable this is:

```
>>>
>>>name="Eric"
>>>age=74
>>>f"Hello, {name}. You are {age}."
```

'Hello, Eric. You are 74.'
It would also be valid to use a capital letter F:

```
>>>
>>>F"Hello, {name}. You are {age}."
```

'Hello, Eric. You are 74.'
Do you love f-strings yet? I hope that, by the end of this article, you'll answer >>>F"Yes!".

Arbitrary Expressions

Because f-strings are evaluated at runtime, you can put any and all valid Python expressions in them. This allows you to do some nifty things.

You could do something pretty straightforward, like this:

```
>>>
>>>f"{2 * 37}"
'74'
```

But you could also call functions. Here's an example:

```
>>>
>>>defto_lowercase(input):
... returninput.lower()

>>>name="Eric Idle"
>>>f"{to_lowercase(name)} is funny."
'eric idle is funny.'
```

You also have the option of calling a method directly:

```
>>>
>>>f"{name.lower()} is funny."
'eric idle is funny.'
```

You could even use objects created from classes with f-strings. Imagine you had the following class:

```
classComedian:
def__init__(self,first_name,last_name,age):
self.first_name=first_name
self.last_name=last_name
self.age=age

def__str__(self):
returnf"{self.first_name}{self.last_name} is {self.age}."

def__repr__(self):
```

returnf"{self.first_name}{self.last_name} is {self.age}. Surprise!"
You'd be able to do this:

```
>>>
>>>new_comedian=Comedian("Eric","Idle","74")
>>>f"{new_comedian}"
'Eric Idle is 74.'
```

Our Other Courses:

1. Artificial Intelligence
2. Data Science
3. Machine Learning
4. DevOps
5. AWS / Azure
6. Internet of Things (IoT)
7. Selenium / Appium
8. ETL Testing
9. Webservices Testing
10. Manual Testing
11. Blueprism
12. Uipath

For Any Queries Fell free to Call: 7730997544 (Teja)