String formatting

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Let's say you have a variable called "name" with your user name in it, and you would then like to print(out a greeting to that user.)

```
name = "John"

print("Hello, %s!" % name)

        output: Hello, John!
```

```
name = "John"

age = 23

print("%s is %d years old." % (name, age))

        output: John is 23 years old.
```

format() function

str.format() is one of the *string formatting methods* in Python3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting.

Using a Single Formatter :

Formatters work by putting in one or more replacement fields and placeholders defined by a pair of curly braces **{ }** into a string and calling the str.format(). The value we wish to put into the placeholders and concatenate with the string passed as parameters into the format function.

Syntax : { } .format(value)

Parameters :
(value) : *Can be an integer, floating point numeric constant, string, characters or even variables.*

Returntype : *Returns a formatted string with the value passed as parameter in the placeholder position.*

```
str = "This article is written in {}"
print (str.format("Python"))

# formatting a string using a numeric constant
print ("Hello, I am {} years old !".format(18))
```

Output :

```
This article is written in Python

Hello, I am 18 years old!
```

Using Multiple Formatters:

Multiple pairs of curly braces can be used while formatting the string. Let's say if another variable substitution is needed in sentence, can be done by adding a second pair of curly braces and passing a second value into the method. Python will replace the placeholders by values in **order.**

Syntax : { } { } .format(value1, value2)

Parameters :

(value1, value2) : Can be integers, floating point numeric constants, strings, characters and even variables. Only difference is, the number of values passed as parameters in format() method must be equal to the number of placeholders created in the string.

Errors and Exceptions :

IndexError : Occurs when string has an extra placeholder and we didn't pass any value for it in the format() method. Python usually assigns the placeholders with default index in order like *0, 1, 2, 3….* to acces the values passed as parameters. So when it encounters a placeholder whose index doesn't have any value passed inside as parameter, it throws IndexError.

```
print ("This is { } { } { } { }" .format("one", "two", "three"))

IndexError: tuple index out of range
```

And if--

```
print ("This is {} {} {} {}"
       .format("one", "two", "three", "four"))
```

Output :

```
This is one two three four
```

Formatters with Positional and Keyword Arguments :

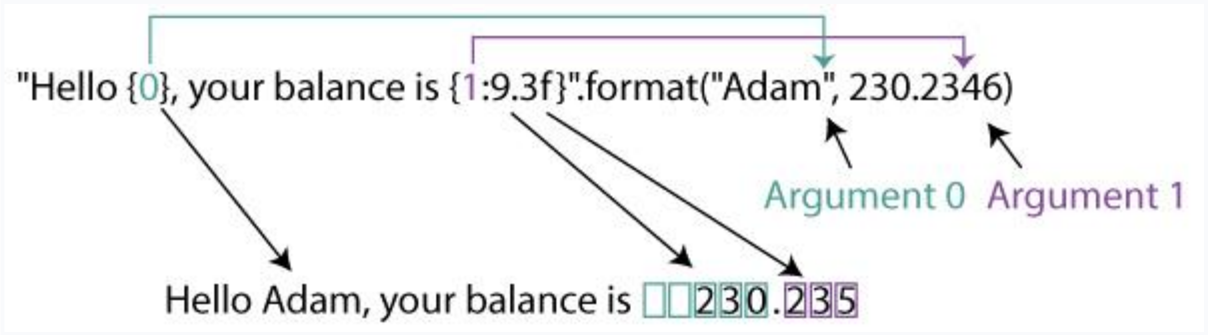
When placeholders **{ }** are empty, Python will replace the values passed through str.format() in order. The values that exist within the str.format() method are essentially **tuple data types** and each individual value contained in the tuple can be called by its index number, which starts with the index number 0. These index numbers can be passes into the curly braces that serve as the placeholders in the original string.

Syntax : *{0} {1}.format(positional_argument, keyword_argument)*

Parameters : *(positional_argument, keyword_argument)*

Positional_argument *can be integers, floating point numeric constants, strings, characters and even variables.*
Keyword_argument *is essentially a variable storing some value, which is passed as parameter.*

For positional arguments


"Hello {0}, your balance is {1:9.3f}".format("Adam", 230.2346)
Argument 0
Argument 1
Hello Adam, your balance is 230.235

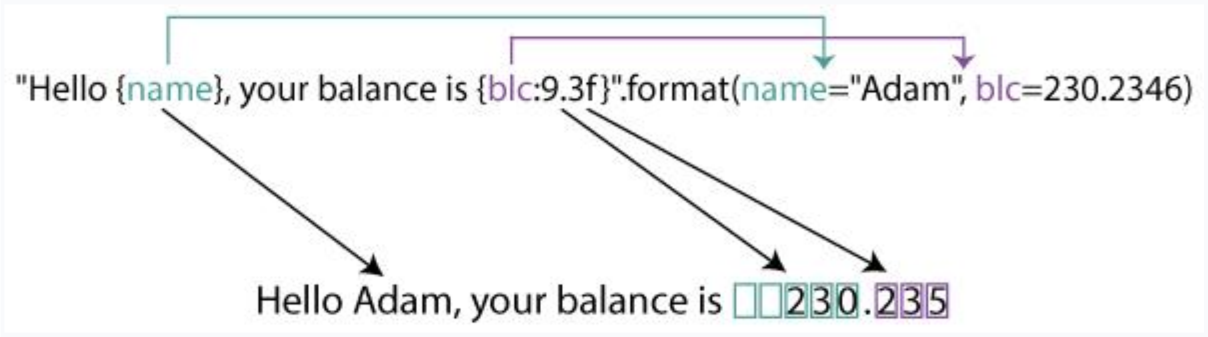
Here, Argument 0 is a string "Adam" and Argument 1 is a floating number 230.2346.

Note: Argument list starts from 0 in Python.

The string "Hello {0}, your balance is {1:9.3f}" is the template string. This contains the format codes for formatting.

The curly braces are just placeholders for the arguments to be placed. In the above example, {0} is placeholder for "Adam" and {1:9.3f} is placeholder for 230.2346.

For keyword arguments


"Hello {name}, your balance is {blc:9.3f}".format(name="Adam", blc=230.2346)
Hello Adam, your balance is 230.235

We've used the same example from above to show the difference between keyword and positional arguments.

Here, instead of just the parameters, we've used a key-value for the parameters. Namely, name="Adam" and blc=230.2346.
Since, these parameters are referenced by their keys as {name} and {blc:9.3f}, they are known as keyword or named arguments.
Internally,

- The placeholder {name} is replaced by the value of name - "Adam". Since, it doesn't contain any other format codes, "Adam" is placed.
- For the argument blc=230.2346, the placeholder {blc:9.3f} is replaced by the value 230.2346. But before replacing it, like previous example, it performs 9.3f operation on it.
  This outputs   230.235. The decimal part is truncated after 3 places and remaining digits are rounded off. Likewise, the total width is assigned 9 leaving two spaces to the left.

Examples:

#named indexes:

txt1 = "My name is {fname}, I'm {age}".format(fname = "John", age = 36)

#numbered indexes:

txt2 = "My name is {0}, I'm {1}".format("John",36)

#empty placeholders:

txt3 = "My name is {}, I'm {}".format("John",36)

print(txt1)

print(txt2)

print(txt3)

output:

```
My name is John, I'm 36
My name is John, I'm 36
My name is John, I'm 36
```