String Formatting Lecture

January 15, 2023

There are three main ways of formatting strings in Python:
1. % formatting
2. using `str.format()` method
3. using f-strings (python >= 3.6)

1 Inserting variables into a string

1.1 Let us start with some ways of inserting variable into a string to be printed that does not use string formatting

1.1.1 Using variables as arguments to `print()` function

[1]:
```
name = "Aditya" # str
num = 101 # int
num_f = 101.11223 # float

print("My name is", name, ". My favourite number is", num, ". Well to be␣
  ↪precise, it is", num_f, ".")
```

```
My name is Aditya . My favourite number is 101 . Well to be precise, it is
101.11223 .
```

Well, writing out `print` in this way forces a space between different arguments. I certainly don't want a space before a period (.).
There is a workaround, but it's certainly not pretty. (`\b` stands for backspace, just like `\n` stands for newline)

[2]:
```
print("My name is", name, "\b. My favourite number is", num, "\b. Well to be␣
  ↪precise, it is", num_f, "\b.")
```

```
My name is Aditya. My favourite number is 101. Well to be precise, it is
101.11223.
```

1.1.2 Using string concatentation

[3]:
```
print("My name is " + name + ". My favourite number is " + str(num) + ". Well␣
  ↪to be precise, it is " + str(num_f) + ".")
```

```
My name is Aditya. My favourite number is 101. Well to be precise, it is
101.11223.
```

There are immediate obvious drawbacks:
1. You need to convert integers and floats to strings. Otherwise, string concatenation doesn't work. Try it!
2. *It looks bad.*

Remember: A good code is always easy on the eyes!

1.2 Let us now try inserting variables using string formatting

1.2.1 Using % formatting

```
[4]: print("My name is %s. My favorite number is %d. Well, to be precise, it is %f."␣
       ↪%(name, num, num_f))

     # Or if there's only one variable
     print("My name is %s." %name)
```

```
My name is Aditya. My favorite number is 101. Well, to be precise, it is
101.112230.
My name is Aditya.
```

First, some comments:
1. As you can see, we use placeholders like `%s` (for strings), `%d` (for integers), `%f` (for floating point numbers).
2. We are using a property of the string class, and not of `print()` function.
This is in contrast to C string formatting where we provide variables as arguments to `printf`, like `printf("My name is %s", name)`.
As a demonstration, see that the following can also be done:

```
[5]: x = "My name is %s. My favourite number is %d. Well to be precise, it is %f."␣
       ↪%(name, num, num_f)
     print(x)
```

```
My name is Aditya. My favourite number is 101. Well to be precise, it is
101.112230.
```

As you can see, string formatting is an easier way of inserting variables into a string. It flows easily as a single statement.

1.2.2 Using `str.format()` method

In this case, we use placeholders `{}` or `{index}` in the string and provide the names of variables as arguments to the `format()` method.
If we use empty `{}` the default indices increase from 0 sequentially.

```
[6]: print("My name is {}. My favorite number is {}. Well, to be precise, it is {}.".
       ↪format(name, num, num_f))
     print("My name is {1}. My favorite number is {0}. Well, to be precise, it is␣
       ↪{2}.".format(num, name, num_f))
```

```
My name is Aditya. My favorite number is 101. Well, to be precise, it is
101.11223.
My name is Aditya. My favorite number is 101. Well, to be precise, it is
101.11223.
```

1.2.3 Using f-strings

f-strings always start with a `f`. These are very simple to use and variable names can be specified inside the string only, using `{variable}`.

```
[7]: print(f"My name is {name}. My favorite number is {num}. Well, to be precise, it␣
      ↪is {num_f}.")
```

```
My name is Aditya. My favorite number is 101. Well, to be precise, it is
101.11223.
```

How to print { and } characters in case of `str.format()` and f-strings?
The answer is using `{{` and `}}`. Try it!

2 Pretty printing using % formatting

Of course, inserting variables into the string isn't the only use of string formatting.
It is mainly used for what we call *pretty printing*.

Here are some use cases:

2.1 Specifying number of digits after decimal for floating point numbers

Suppose we want to print `num_f` only upto two decimal places.

We use `%.yf` instead of `%f`, where, `y` denotes the required number of digits after decimal.

```
[8]: print("My name is %s. My favorite number is %d. Well, to be precise, it is %.2f.
      ↪" %(name, num, num_f))
```

```
My name is Aditya. My favorite number is 101. Well, to be precise, it is 101.11.
```

Side Note: I want to mention that integers can be implicity converted to floating points just by using `%f` where `%d` should have been.
(Although, floating point numbers cannot be converted to integers implicitly).
An example is:

```
[9]: print("My name is %s. My favorite number is %.3f. Well, to be precise, it is %.
      ↪2f." %(name, num, num_f))
```

```
My name is Aditya. My favorite number is 101.000. Well, to be precise, it is
101.11.
```

2.2 Inserting padding

We use `%xs`, `%xd`, `%xf` instead of `%s`, `%d`, `%f`, where `x` denotes the number of characters wide each field must **atleast** be.

This is often important when printing a table. Let us see an example:

```
[10]: names = ["Aditya", "Rohan", "Angel", "Calvin"]
      nums = [101, 89814, 21, 1]
      nums_f = [101.11223, 89814.33123, 21.15983, 1.93012]

      for i in range(4):
          print("%s %d %f" %(names[i], nums[i], nums_f[i]))
```

```
Aditya 101 101.112230
Rohan 89814 89814.331230
Angel 21 21.159830
Calvin 1 1.930120
```

```
[11]: # Let us now try using padding
      for i in range(4):
          print("%8s %6d %15f" %(names[i], nums[i], nums_f[i]))
```

```
  Aditya    101      101.112230
   Rohan  89814    89814.331230
   Angel     21       21.159830
  Calvin      1        1.930120
```

2.3 Specifying number of characters in the string variable

For very long strings, specifying a number after a dot (like `%.8s`), terminates the string after that many characters.
For example,

```
[12]: names = ["Aditya Dutta of IISER Kolkata", "Rohan", "Angel", "Calvin"]
```

```
[13]: for i in range(4):
          print("%8s %6d %15f" %(names[i], nums[i], nums_f[i]))
```

```
Aditya Dutta of IISER Kolkata    101      101.112230
   Rohan  89814    89814.331230
   Angel     21       21.159830
  Calvin      1        1.930120
```

```
[14]: # Let us now print only upto 8 characters
      for i in range(4):
          print("%8.8s %6d %15f" %(names[i], nums[i], nums_f[i]))
```

```
Aditya D    101      101.112230
   Rohan  89814    89814.331230
   Angel     21       21.159830
  Calvin      1        1.930120
```

2.4 Conclusion

1. In case of `%x.ys`, `x` denotes the padding and `y` denotes number of characters after which string is to be terminated.
2. In case of `%x.yf`, `x` denotes the padding and `y` denotes the number of digits after decimal.
3. In case of `%x.yd`, `x` denotes the padding. Find out what `y` denotes.

[15]:
```
for i in range(4):
    print("%8.8s %6.3d %10.2f" %(names[i], nums[i], nums_f[i]))
```

```
Aditya D    101     101.11
   Rohan  89814   89814.33
   Angel    021      21.16
  Calvin    001       1.93
```

3 Pretty printing using `str.format()`

We just use a colon `:` and give `x.y` after that, like `{:x.y}` or `{index:x.y}`.

[16]:
```
for i in range(4):
    print("{:8.8} {:6} {:10.2f}".format(names[i], nums[i], nums_f[i]))
```

```
Aditya D    101     101.11
Rohan     89814   89814.33
Angel        21      21.16
Calvin        1       1.93
```

[17]:
```
for i in range(4):
    print("{1:8.8} {0:6} {2:10.2f}".format(nums[i], names[i], nums_f[i]))
```

```
Aditya D    101     101.11
Rohan     89814   89814.33
Angel        21      21.16
Calvin        1       1.93
```

Note: In case of float, we used `f` at the end i.e. `{:x.yf}` or `{index:x.yf}`. By default, it will use scientific notation instead. Try for yourself.

Note: I would also like to emphasize that specifying a number after dot in case of integers gives an error when using `str.format()` or f-strings (unlike when using `%` formatting).
Again, specifying an `f` at the end will convert the integer to float implicity, due to which `.y` works, but now it just denotes the number of digits after decimal and not what is used to when using `%.yd` formatting.

4 Pretty printing using f-strings

Just like `str.format()` method, we use a colon `:` and give `x.y` after that, like `{variable:x.y}` .

[18]:
```
for i in range(4):
    print(f"{names[i]:8.8} {nums[i]:6} {nums_f[i]:10.2f}")
```

```
Aditya D    101      101.11
Rohan     89814    89814.33
Angel        21       21.16
Calvin        1        1.93
```

5 Takeaway

We saw three methods of formatting strings:
1. Using `%`
2. `str.format()` method
3. f-strings

5.1 Format

The general format for each is:

5.1.1 Using %

`"String containing placeholders like %x.ys %x.yd %x.yf" %(tuple of variables in the order they occur in the string)`

In case of single variable,
`"String containing placeholder like %x.ys OR %x.yd OR %x.yf" %variable`

5.1.2 Using `str.format()`

`"String containing placeholders as {:x.y} or {index:x.y}".format(variables as arguments)`

5.1.3 Using f-strings

`f"String containing placeholders as {variables:x.y}"`

Note: As we discussed, when using `str.format()` or f-strings, write `f` at the end after `y` in case of float type to get result in floating points and not scientific notation.

5.2 `x` and `y`

`x` always denotes the padding or the number of characters wide that field must **atleast** be.

`y` denotes:
1. For strings: The number of characters after which the string is terminated.
2. For floating points: The number of digits after decimal.
3. For integers, it works when using `%` but gives an error when using `str.format()` or f-strings.

5.3 `str.format()` and f-strings are much more powerful

Lastly, I want to mention that `str.format()` method and f-strings are much more powerful than what I have demonstrated.
Giving `x.y` or even `f` as specification (as we did for floating point numbers), is a small subset of

specifications that can be given for string formatting.
Read more about them on python’s official tutorial.