Introduction

There exists two ways to print a variable content:

1. Use it as an argument of `print`.

   ```
   var = 5
   print("Var: ", var)
   ```

   Then, the `__print__` method of instances is called.

2. Insert it in a string with special formatting option, and print this string.

Method 1: no formatting

This method is fully compliant with the **unpacking** technic:

In [1]:
```
data = "abcdefg"
print(*data)
print(*data, sep="/")
```

```
a b c d e f g
a/b/c/d/e/f/g
```

In [2]:
```
data = [1, 2, 3, 4]
print(*data)
print(*data, end="/")
```

```
1 2 3 4
1 2 3 4/
```

Method 2: advanced formatting

Advanced formatting is interesting in a scientific approach because it presents the variable value according to its type. One can see this tutorial for specific use cases. Most of them are covered hereafter.

Key idea

Variable name is enclosed in curly brackets `{}` and a prefix `f` is added in front of the string.

In [3]:
```
var = 35.123456
sentence = f"Value of 'var' is {var}"
print(sentence)
```

```
Value of 'var' is 35.123456
```

Symbols `<`, `>` and `^` produces text-alignment: left, right and center. If any character precedes one of these symbols, the character is used in a **padding way**.

Floats

If the variable is to be printed as a float, an additional formatting using the 'f' letter is used inside the curly brackets. One can specify:

- Number of decimal figures
- Minimal number of caracters

In [4]:
```
var = 35.123456
print(f"{var:<9.2f}")
print(f"{var:_<9.2f}")    # padding
print(f"{var:_>9.2f}")    # padding
print(f"{var:_^9.2f}")    # padding
```

```
35.12
35.12____
____35.12
__35.12__
```

Scientific notation

Similar to float values representation, yet with letter 'e':

In [5]:
```
var = 35.123456
print(f"{var:_>9.2e}")
```

```
_3.51e+01
```

Percentages

Percentage mode involves the symbol '%': what is printed is the product of the variable by 100.

In [6]:
```
var = 0.35123456
print(f"{var:_>9.2%}")
```

```
___35.12%
```

Int

The total number of characters is specified.

In [7]:
```
var = 12356
print(f"{var:0>9}")
```

```
000012356
```

Beware! For Python an integer that ends with `.` is a float!

Other ways to specify arguments

Positional arguments

In [8]:
```
data = range(5)
print("third value = {2}".format(*data))
```

```
third value = 2
```

Named arguments

In [9]:
```
data = {"key1": 0, "key2": 1}
print("'key1' = {key1}".format(**data))
```

```
'key1' = 0
```

Advanced: call `str` or `repr`

By default, the `__format__` method of instances is called. One can change to use:

- `str`: `{var!s}` (overload of `__str__`)
- `repr`: `{var!r}` (overload of `__repr__`)

In [10]:

```
class Fake(float):

    def __repr__(self):
        return "This is my Float (repr)\n"

    def __str__(self):
        return "This is my Float (str)\n"

    def __format__(self, *args, **kwargs):
        return super().__format__(f"{123456.32145:2.3f}")

var = Fake()
print(f"This is formatted: {var}")
print(f"This is printed: {var!s}")
print(f"This is represented: {var!r}")
```

```
This is formatted:
0.0
This is printed: This is my Float (str)

This is represented: This is my Float (repr)
```