The `print()` function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

Syntax:

```
print(*values, sep= seperator, end = end, file = file, flush = flush)
```

where,

- `value(s)` : Any object, * indicats as many as you like. Everthing will be convered to `string` before printed.
- `sep='separator'` : **Optional.** Specify how to separate the objects, if there is more than one. Default is `' '`
- `end='end'` : **Optional.** Specify what to print at the end. Default is `'\n'` (line feed)
- `file` : **Optional.** An object with a write method. Default is `sys.stdout`
- `flush` : **Optional.** A Boolean, specifying if the output is flushed (True) or buffered (False). Default is `False`

Note: `sep`, `end`, `file`, and `flush` are keyword arguments. If you want to use sep argument, you have to use:

```
print(*values, sep = 'separator', end = 'end')
```

Examples:

In [1]:
```
message = "Hello World"
#print the string message
print(message)
```

```
Hello World
```

In [2]:
```
#We can directly used string "Hello World inside print() without using variable message"
print("Hello World")
```

```
Hello World
```

In [3]:
```
#Printing two values in same print()
print("Hello!", "how are you?")
```

```
Hello! how are you?
```

In [4]:
```
#printing complex datatype such as tuples
x = ("apple", "banana", "cherry")
print(x)
```

```
('apple', 'banana', 'cherry')
```

Example of print using `sep`

In [5]:
```
#without using sep
print("Hello","This","is","fun")
print("-"*25)
#Using sep
print("Hello","This","is","fun", sep="-")
```

```
Hello This is fun
-------------------------
Hello-This-is-fun
```

Example of print using `end`

In [6]:
```python
#Example of print using without end
for i in range(5):
    print(i)
print("-"*25)
#Example of print using end
for i in range(5):
    print(i, end="-")
```

```
0
1
2
3
4
-------------------------
0-1-2-3-4-
```

String formatting in print:

% operator: also called as “Old Style” String Formatting. Strings in Python have a unique built-in operation that can be accessed with the `%` operator. This lets you do simple positional formatting very easily. If you’ve ever worked with a printf-style function in C, you’ll recognize how this works instantly.

Note: It is recommended to avoid using % operator as string literals are more preferred way since python 3

Here’s a simple example:

In [7]:
```python
name = "abcd"
print('Hello, %s' % name)
age = 25
print('abcd is %d years old' % age)
```

```
Hello, abcd
abcd is 25 years old
```

String Formatting (str.format): also called as "New Style" String formatting.This “new style” string formatting gets rid of the %-operator special syntax and makes the syntax for string formatting more regular. Formatting is now handled by calling `.format()` on a string object.

You can use format() to do simple positional formatting, just like you could with “old style” formatting:

Examples:

In [8]:
```python
name = "abcd"
print('Hello, {}'.format(name))
age = 25
print('abcd is {} years old'.format(age))
```

```
Hello, abcd
abcd is 25 years old
```

Formatted String Literals or `f-strings` : If your are using Python 3.6+, string f-strings are the recommended way to format strings.

A formatted string literal or f-string is a string literal that is prefixed with `f` or `F` . These strings may contain replacement fields, which are expressions delimited by curly braces `{}` . While other string literals always have a constant value, formatted strings are really expressions evaluated at run time.

This are also called **String Interpolation**

Example:

```
In [9]: name = "abcd"
        print(f'Hello, {name}')
        age = 25
        print(f'abcd is {age} years old')
```

```
Hello, abcd
abcd is 25 years old
```

Formatting the digits in python

```
In [10]: #Adding thousands separator
         a = 10000000
         print(f"{a:,}")
```

```
10,000,000
```

```
In [11]: #Rounding
         a = 3.1415926
         f"{a:.2f}"
```

```
Out[11]: '3.14'
```

```
In [12]: #Showing as Percentage
         a = 0.816562
         print(f"{a:.2%}")
```

```
81.66%
```

```
In [13]: a = 11
         print(f"{a:11d}")
```

```
         11
```

Below is number formatting table:

Number	Format	Output	description
3.1415926	{:.2f}	3.14	Format float 2 decimal places
3.1415926	{:+.2f}	+3.14	Format float 2 decimal places with sign
-1	{:+.2f}	-1.00	Format float 2 decimal places with sign
2.71828	{:.0f}	3	Format float with no decimal places
4	{:0>2d}	04	Pad number with zeros (left padding, width 2)
4	{:x<4d}	4xxx	Pad number with x's (right padding, width 4)
10	{:x<4d}	10xx	Pad number with x's (right padding, width 4)
1000000	{:,}	1,000,000	Number format with comma separator
0.35	{:.2%}	35.00%	Format percentage
1000000000	{:.2e}	1.00e+09	Exponent notation
11	{:11d}	11	Right-aligned (default, width 10)
11	{:<11d}	11	Left-aligned (width 10)
11	{:^11d}	11	Center aligned (width 10)