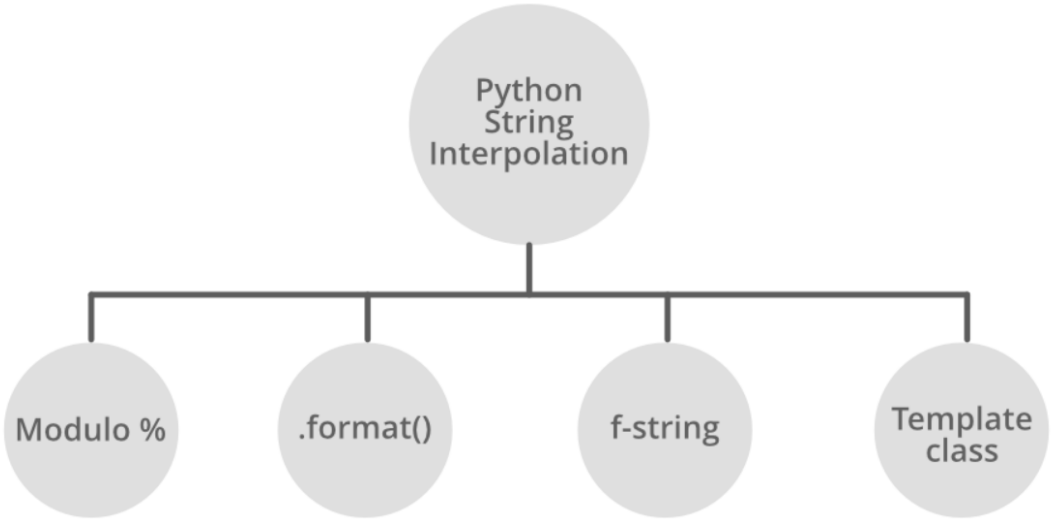
What is String Interpolation?

Ans:- It is the process of **inserting variables or expressions inside strings** instead of concatenating them manually.



1. %-Formatting (C-style formatting)

Python allows string interpolation using the % operator, similar to printf in C. You can insert variables into strings without concatenation.

Example:-

```
name = "Ankit"

platform = "Python"


# Single substitution

print("Welcome to %s!" % platform)


# Multiple substitutions

print("%s is learning %s." % (name, platform))
```

Explanation:

- %s is a placeholder for a string.
- %d can be used for integers, %f for floating-point numbers, etc.
- Multiple variables are passed as a tuple after the % operator.

Why use it?
It avoids long concatenations like "Hello " + name + "!" and keeps the string clean, especially with multiple variables.

2. str.format() Method

Introduced in Python 2.6/3.0, the format() method allows more control over string formatting. Placeholders are denoted by {} inside the string.

Example:-

first = "Ankit"

second = "Python"

Simple substitution

print("Hello {}, welcome to {}.".format(first, second))

Using named placeholders

print("{user} is mastering {language}.".format(user=first, language=second))

Changing order of placeholders

print("{language} is loved by {user}.".format(user=first, language=second))

Explanation:

- {} are placeholders that can be positional or named.
- Named placeholders allow flexibility in reordering without changing the arguments.

Why use it?
It's more readable than %-formatting and works well when dealing with multiple variables.

3. F-strings (Literal String Interpolation)

Introduced in **Python 3.6**, f-strings provide a concise and modern way to interpolate strings. Prefix a string with f and use {} to embed variables or expressions directly.

```
user = "Ankit"

language = "Python"

# Basic interpolation

print(f"Hello {user}, welcome to {language}!")

# Inline arithmetic

x = 5

y = 7

print(f"{x} + {y} = {x + y}")
```

Explanation:

- Variables and expressions can be directly written inside {}.
- Supports inline calculations, function calls, and even formatting options like {value:.2f}.

Why use it?
It’s the most readable and efficient method in modern Python, especially when dealing with multiple variables and expressions.

4. String Template Class

The **Template class** from Python’s string module provides a simple way to substitute variables in strings. Placeholders are prefixed with $, making it safe and readable for basic templates.

```
from string import Template

greeting = Template("Hello $user! Welcome to $platform.")

print(greeting.substitute(user="Ankit", platform="Python"))

# Escaping $ character

price_template = Template("The total cost is $$100.")

print(price_template.substitute())
```

Explanation:

- $variable denotes a placeholder.
- substitute() replaces placeholders with actual values.
- $$ allows escaping the $ symbol.

Why use it?
Templates are useful when working with strings provided by users or external sources, as they provide a safe and readable syntax.

SUMMARISATION:-

Method	Placeholder	Python Version	Notes
`%`-Formatting	`%s`, `%d`, `%f`	All versions	Similar to C `printf`, older style
`str.format()`	`{}` or `{name}`	2.6+ / 3.0+	Flexible, allows reordering and naming
F-strings	`{}` inside `f""`	3.6+	Most modern, supports expressions and formatting
Template	`$variable`	2.4+	Safe for external input, simple syntax