Chapter 18: PRINT

(Class 9 Artificial Intelligence)

Introduction

In programming, one of the most essential operations is **displaying output to the user**. This is done using the **`print()` function** in Python. The `print()` function allows a programmer to show messages, values of variables, results of calculations, and much more to the screen. Whether you're debugging code or building user-friendly software, mastering the use of `print()` is fundamental.

In this chapter, you will learn what the `print()` function is, how it works, and how to use it effectively with various data types, formatting options, and escape characters.

18.1 What is the `print()` Function?

The `print()` function in Python is used to **display output on the screen**. It is a **built-in function**, meaning it's available without any import or special declaration.

Syntax:

```
print(object(s), sep=' ', end='\n', file=sys.stdout, flush=False)
```

Parameters:

- **object(s)** – Any number of objects to be printed. Separated by commas.
- **sep** – Optional. Separator between objects. Default is a space `' '`.
- **end** – Optional. String appended after the last value. Default is newline `'\n'`.
- **file** – Optional. An object with a write method. Default is `sys.stdout`.
- **flush** – Optional. Whether to forcibly flush the stream. Default is `False`.

18.2 Printing Strings

Strings are text enclosed in **single quotes (' ')**, **double quotes (" ")**, or **triple quotes (''' ''' or """ """)**.

Example:

```
print("Hello, World!")
```

Output:

```
Hello, World!
```

18.3 Printing Numbers and Expressions

You can use `print()` to display numbers and even solve arithmetic expressions directly.

Example:

```
print(10)
print(5 + 3)
```

Output:

```
10
8
```

18.4 Printing Multiple Values

The `print()` function can take **multiple arguments**, separated by commas.

Example:

```
a = 5
b = 10
print("The values are", a, "and", b)
```

Output:

```
The values are 5 and 10
```

18.5 The `sep` Parameter

The `sep` parameter controls what is printed **between multiple items**.

Example:

```
print("10", "20", "30", sep="-")
```

Output:

```
10-20-30
```

This is useful when printing dates, times, or IDs with specific formatting.

18.6 The `end` Parameter

The `end` parameter controls what is printed **after the statement ends**. By default, it's a **newline (`\n`)**, but you can change it.

Example:

```
print("Hello", end=" ")
print("World")
```

Output:

```
Hello World
```

18.7 Escape Characters

Escape characters start with a **backslash (\)** and allow you to include special characters in strings.

Escape Sequence	Description
\n	New line
\t	Tab space
\\	Backslash
\'	Single quote
\"	Double quote

Example:

```
print("Line1\nLine2")
print("She said, \"Hello!\"")
```

Output:

```
Line1
Line2
She said, "Hello!"
```

18.8 Printing Variables

You can use variables with the `print()` function to show their values.

Example:

```
name = "Ravi"
age = 14
print("Name:", name)
print("Age:", age)
```

Output:

```
Name: Ravi
Age: 14
```

18.9 Printing Using f-strings (Formatted Strings)

Introduced in Python 3.6, **f-strings** allow you to embed variables directly inside strings.

Example:

```
name = "Anita"
score = 95
print(f"{name} scored {score} marks.")
```

Output:

```
Anita scored 95 marks.
```

F-strings make code more readable and concise.

18.10 Printing with `.format()`

Another way to insert values into a string is by using the `.format()` method.

Example:

```
print("My name is {} and I am {} years old".format("Rahul", 15))
```

Output:

```
My name is Rahul and I am 15 years old
```

18.11 Printing with Concatenation

You can also use the **+ operator** to join strings.

Example:

```
name = "Aman"
print("Hello, " + name)
```

Output:

```
Hello, Aman
```

Note: All items must be strings when using `+`. You must convert numbers using `str()`.

18.12 Common Errors

❌ Mixing string with numbers:

```
age = 14
print("Age is " + age)  # Error!
```

✅ Fix:

```
print("Age is " + str(age))
```

Summary

- The `print()` function is used to **display output** on the screen.
- You can print **strings**, **numbers**, **variables**, or **expressions**.
- Use **`sep`** to change separators, and **`end`** to change line endings.
- Escape sequences like `\n` and `\t` help format your output.
- `f-strings` and `.format()` make printing dynamic messages easier.
- Always convert **non-string values** to string when using concatenation (`+`).
