1. Introduction to the print() function in Python.

The print() function in Python is used to output messages or data to the standard output, usually the console. It is one of the most fundamental and frequently used functions for displaying information during program execution.

Basic Usage

The simplest use is to pass a string or any other object(s) as arguments:

python

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

print(123)

print([1, 2, 3])
```

The print() function converts objects into strings and writes them to the screen.

Syntax

python

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

- *objects: One or more objects to print, separated by commas.
- sep: String inserted between objects (default is a space).
- end: String appended at the end (default is newline \n).
- file: Output stream (default is standard output).
- flush: Whether to forcibly flush the stream buffer.

Examples

- Printing multiple items with custom separator:

python

```
print("Hello", "World", sep=", ")
# Output: Hello, World
```

- Avoiding new line after print:

python

```
print("Hello", end=" ")
print("World")
# Output: Hello World
```

- Printing empty line:

python

```
print()
```

Characteristics

- Automatically adds a newline at the end unless overridden with end.
- Can print any data type by converting it to string.
- Useful for debugging, user interaction ,and displaying result.

2. Formatting outputs using f-strings and format().

In Python, outputs can be formatted for clarity and precision using f-strings (formatted string literals) and the format() method. Both techniques allow you to embed variables, control alignment, and specify numeric precision directly within strings.

f-strings

F-strings, introduced in Python 3.6, use the syntax f"text {expression}".

- Variables and expressions are inserted using curly braces {}.
- Supports format specifiers, e.g., :.2f for two decimal places, :>8 for right alignment.

Example:

python

```
name = "Alice"
```

score = 95.5

print(f"Hello, {name}. Your score is {score:.2f}.")

 # Output: Hello, Alice. Your score is 95.50.

You can use modifiers for alignment and width:

python

val = 3.14159

print(f"|{val:10.2f}|") *# Right align in 10 spaces*

print(f"|{val:<10.2f}|") *# Left align in 10 spaces*

Multiline f-strings are possible with triple quotes for longer structured outputs.

format() method

The .format() method provides formatting in older and current Python versions.

- Placeholders {} are replaced by arguments in .format().
- Supports positional and keyword arguments, as well as format specifiers.

Example:

python

name = "Bob"

score = 87.42

print("Hello, {}. Your score is {:.1f}.".format(name, score))

 # Output: Hello, Bob. Your score is 87.4.

Alignment and width:

python

val = 3.14159

print("{:10.2f}".format(val)) *# 10 positions, 2 decimals*

print("{:<10.2f}".format(val)) *# Left aligned*

Both methods streamline output formatting for readable, organized results. F-strings are preferred for their clearer and more concise syntax.

3. Using the input() function to read user input from the keyboard

The input() function in Python is used to read user input from the keyboard during program execution. When input() is called, the program pauses and waits for the user to type something and press Enter.

Syntax

python

variable = input("Prompt message: ")

- The message (prompt) inside the quotes is shown to the user.
- The returned value is always a string. You can convert it to other data types if needed.

Examples

1. Basic input

python

name = input("What is your name? ")

print("Hello,", name)

 # Output: What is your name? John

 # Hello, John

2. Input as integer

python

num = int(input("Enter a number: "))

print("Number squared is:", num * num)

 # Output: (user enters 5) Number squared is: 25

3. Multiple inputs in one line

python

```
x, y = input("Enter two numbers: ").split()
print("First number:", x)
print("Second number:", y)
```

4. Input for arithmetic operations

python

```
n1 = int(input("First Number: "))
n2 = int(input("Second Number: "))
print("Sum:", n1 + n2)
```

Important Notes

- All inputs are read as strings. Use int() or float() to convert them if you need numeric input.
- The prompt are read as string healps guide the user on what to enter.

The input() function is essential for interactive Python programs, enabling real-time use interaction.

4. Converting user input into different data types (e.g., int, float, etc.).

Converting user input into different data types in Python is a common need since the input() function returns everything as a string by default. You must explicitly convert (cast) the input to the desired data type.

Common Conversions

- To integer:

python

```
num = int(input("Enter an integer: "))
```

Converts the input string to an integer. If the user enters a non-numeric string, it raises a ValueError.

- To float:

python

value = float(input("Enter a float number: "))

Converts the input string to a floating-point number.

- To string:

python

text = input("Enter some text: ")

No conversion needed, input is already a string.

- To Boolean:
  You can convert strings like "True" or "False" explicitly using:

python

bool_val = input("Enter True or False: ") == "True"

Handling invalid inputs with try-except

To prevent your program from crashing on invalid input, you can use a try-except block:

python

```
try:
    num = int(input("Enter an integer: "))
except ValueError:
    print("Invalid input, please enter a valid integer.")
```

Example: Read and convert input safely

python

```
while True:
    try:
        num = int(input("Enter an integer: "))
```

```
        break
    except ValueError:
        print("That's not a valid integer. Try again.")
print("You entered:", num)
```

Summary

- Use int(), float(), str() and other type constructors to convert input.
- Always validate or handle execeptions to deal with invalid inputs gracefully.
- Python also performs implicit conversion automatically in some expressions involving mixed types

5. Opening files in different modes ('r', 'w', 'a', 'r+', 'w+')

Python provides several file modes:

- r: Reads an existing file; file must exist.
- w: Writes to a file; overwrites existing content or creates a new file.
- a: Appends data to the end of a file; creates the file if it doesn’t exist.
- r+: Reads and writes; file must exist.
- w+: Reads and writes but overwrites the file or creates a new one.

---

6) Question 6:

What is the purpose of the open() function in Python?

Answer:
The open() function is used to create, read, write, or modify files. It returns a file object through which all file operations are performed.

---

7) Question 7:

Why is it important to close a file using the close() method?

Answer:
Closing a file ensures that all data is saved and system resources are released. It prevents file corruption and memory leaks.

---

8) Question 8:

Explain read(), readline(), and readlines().

Answer:

- read(): Reads the entire file as a single string.
- readline(): Reads one line at a time.
- readlines(): Reads all lines and returns them as a list.

---

9) Question 9:

What is the difference between write() and writelines()?

Answer:

- write() writes a single string to a file.
- writelines() writes multiple strings (a list) without automatically adding newline characters.

---

10) Question 10:

What are exceptions in Python, and how are they handled?

Answer:
Exceptions are runtime errors. They are handled using:

- try: Code that might cause an error.
- except: Handles the error.
- finally: Executes regardless of errors, usually for cleanup.

---

11) Question 11:

What are multiple exceptions and custom exceptions?

Answer:

- Multiple exceptions: Using different except blocks to handle different errors.
- Custom exceptions: User-defined errors created by extending the Exception class.

---

12) Question 12:

Explain classes, objects, attributes, and methods.

Answer:

- Class: A blueprint for objects.
- Object: An instance of a class.
- Attributes: Variables that hold data.
- Methods: Functions inside a class that operate on its data.

---

13) Question 13:

What is the difference between local and global variables?

Answer:

- Local variables exist inside functions or methods.
- Global variables are defined outside functions and accessible throughout the program.

---

14) Question 14:

Describe the types of inheritance in Python.

Answer:

- Single: One parent → one child.

- Multilevel: Parent → Child → Grandchild.
- Multiple: Child with more than one parent.
- Hierarchical: One parent, multiple children.
- Hybrid: Combination of multiple inheritance forms.

---

15) Question 15:

What is the use of the super() function?

Answer:
super() allows a child class to access methods and attributes of its parent class, especially useful during method overriding.

---

16) Question 16:

What is method overloading?

Answer:
Method overloading means having multiple methods with the same name but different parameters. Python simulates this using default or variable-length arguments.

---

17) Question 17:

What is method overriding?

Answer:
Method overriding occurs when a child class redefines a method from its parent to modify or extend its behavior.

---

18) Question 18:

What are SQLite3 and PyMySQL used for?

Answer:

- SQLite3 is a lightweight, file-based database for local storage.

- PyMySQL connects Python programs to a MySQL server. Both allow executing SQL commands through Python.

---

19) Question 19:

What is the difference between re.search() and re.match()?

Answer:

- re.match() checks for a pattern at the beginning of a string.
- re.search() looks for the pattern anywhere in the string.