Print function and String Formatting

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology
Chennnai Campus

premanand.s@vit.ac.in

October 23, 2024

Introduction to the `print()` Function

- The `print()` function outputs data to the standard output (console).
- You can pass multiple arguments, and by default, they are separated by a space.
- Each call to `print()` automatically adds a newline unless modified.
- The function can handle different data types without explicit conversion (e.g., integers, strings, lists).

Example (Python Code)

```
# Printing strings
print("Hello, World!")
# Printing multiple arguments
print("This is", "Python")
# Using variables
name = "Python"
age = 1991
print("Name:", name, "Age:", age)
```

Printing Variables

- `print()` can handle multiple variables and automatically convert them to strings.
- No need for explicit concatenation with the + operator when using commas.

Example (Python Code)

```
# Defining variables
name = "Premanand"
age = 38
# Printing multiple variables
print("Name:", name, "Age:", age)
# String concatenation using '+'
print("Name: " + name + ", Age: " + str(age))
```

Customizing the `print()` Function

- The `sep` parameter defines what to use between multiple arguments.
- Default value: `sep=' '` (space), but it can be modified.

Example (Python Code)

```
# Using a custom separator
print("apple", "banana", "cherry", sep=", ")

# Using no separator (empty string)
print("apple", "banana", "cherry", sep="")
```

Controlling the End of a Print Statement

- By default, `print()` adds a newline (`end='\n'`) after each call.
- The `end` parameter can be customized to prevent newline or append other characters.

Example (Python Code)

```
# Default behavior: adds a newline after printing
print("Hello")
print("World")

# Customizing the end parameter
print("Hello", end=" ")
print("World")

# Using custom characters at the end
print("Loading", end="... Done!")
```

Escape Characters in `print()`

- Escape characters are used to represent special characters within strings.
- Common escape sequences include:
  - `\n` for newline
  - `\t` for tab
  - `\\` for a literal backslash

Example (Python Code)

```
# Newline
print("Hello\nWorld")
# Tab
print("Item\tPrice")
# Literal backslash
print("This is a backslash: \\")
```

Escape Characters in ‘print()‘

- Common escape characters include quotes too
- You can use escape sequences for special characters:

Example (Python Code)

```
print("Quotes: \" and \'")  # Quotes
```

String Formatting with ‘format()‘

- The `format()` method allows flexible string formatting without concatenation.
- Positional and keyword arguments:

Example (Python Code)

```
print("My name is {} and I am {} years old".
                                format(name, age))
print("My name is {name} and I am {age} years old".format
              (name="Premanand", age=38))
```

Advanced String Formatting with ‘format()‘

- Width, alignment, and precision:

Example (Python Code)

```
# Left-aligned, right-aligned, and centered text
print("{:<10} is left-aligned".format("Text"))
print("{:>10} is right-aligned".format("Text"))
print("{:^10} is centered".format("Text"))

# Floating-point precision
print("Pi is approximately {:.2f}".format(3.1415926535))
```

F-Strings in Python (Python 3.6+)

- F-Strings allow embedding expressions and variables directly within strings.
- F-Strings provide a concise way to format strings:

Example (Python Code)

```
print(f"My name is {name} and I am {age} years old")
print(f"Pi rounded to 2 decimal places is
                                    {3.1415926535:.2f}")
```

String Formatting for Different Data Types

- Integers, hexadecimals, and binary:

Example (Python Code)

```
print(f"The number 255 in hexadecimal is {255:#x}")
print(f"The number 15 in binary is {15:#b}")

Output:
The number 255 in hexadecimal is 0xff
The number 15 in binary is 0b1111
```

Multi-line Printing

- You can print multi-line text using triple quotes:

Example (Python Code)

```
print("""This is a multi-line
string using triple quotes""")
```

Printing with ‘end‘ for Inline Output

- You can use end to control how the printed output behaves:

Example (Python Code)

```
for i in range(5):
    print(i, end=", ")  # Prints all numbers in a single line
```

Printing Complex Data Structures

- Printing complex data structures like lists and dictionaries:

Example (Python Code)

```
my_list = [1, 2, 3, 4]
print(f"My list: {my_list}")

my_dict = {"name": "Python", "age": 1991}
print(f"My dictionary: {my_dict}")
```

Handling Encoding in 'print()'

- Python 3's `print()` supports Unicode by default, allowing you to print characters from many languages.
- Unicode escape sequences can be used to print special symbols.
- Useful for printing special mathematical symbols, foreign language characters, emojis, etc.

Example (Python Code)

```
print("Greek Omega: \u03A9") # Greek Omega character ()

print("Currency symbol: \u20AC") # Euro symbol (€)

print("Heart Emoji: \u2665") # Heart emoji ()

print("Chinese Character: \u4F60") # Chinese character ()

print("Smiley Emoji: \U0001F60A") # Printing a smiley emoji
```

Printing to a File

- This writes the output to `output.txt` instead of the console.
- You can redirect output to a file using the `file` parameter:

Example (Python Code)

```
with open('output.txt', 'w') as f:
    print("Writing to a file", file=f)
```

Conclusion

- `print()` is a versatile function for debugging and user interaction.
- Various formatting methods (`format()`, F-Strings) allow control over output.
- Advanced features (alignment, precision, escape characters) make it powerful for structured output.