Chapter 5: Formatting Text Precisely Tutorial

A Comprehensive Guide to Python Text Formatting Techniques

Tutorial by Ayes Chinmay

July 25, 2025

Contents

1 Introduction

This tutorial explores Python's text formatting techniques, covering the percent sign operator (%), the global format function, and the string format method. These tools allow precise control over how data is displayed, including alignment, precision, and number formatting. This tutorial is based on the structure of Chapter 5, "Formatting Text Precisely," and provides detailed explanations, Python code examples, and practical applications.

2 Formatting with the Percent Sign Operator (%)

The percent sign operator (%), also known as the string formatting operator, is an older method for formatting strings in Python. It uses format specifiers to define how values are inserted into a string.

```
# Example: Basic percent sign formatting
name = "Alice"
age = 25
print("Name: %s, Age: %d" % (name, age))
# Output: Name: Alice, Age: 25
```

The %s specifier formats strings, while %d formats integers. The values are provided in a tuple after the % operator.

3 Percent Sign (%) Format Specifiers

Format specifiers define the type and style of the data. Common specifiers include:

- %s: String
- %d: Integer
- %f: Floating-point number
- %x: Hexadecimal (lowercase)
- %o: Octal

```
# Example: Using different specifiers
value = 42
print("Integer: %d, Hex: %x, Octal: %o" % (value, value, value))
# Output: Integer: 42, Hex: 2a, Octal: 52
```

4 Percent Sign (%) Variable-Length Print Fields

You can specify the width of the print field using a number between % and the specifier. A negative number left-justifies the output.

```
# Example: Variable-length fields
print("Name: %-10s Age: %5d" % ("Alice", 25))
# Output: Name: Alice      Age:    25
```

5 The Global format Function

The format function, introduced in Python 2.6, provides a more flexible way to format strings. It uses placeholders {} and can be customized with format specifications.

```
# Example: Using the format function
print(format(42, "d"), format(42, "x"), format(3.14159, ".2f"))
# Output: 42 2a 3.14
```

6 Introduction to the format Method

The string format method, introduced in Python 3, allows formatting by replacing placeholders {} with values. It is more readable than the % operator.

```
# Example: Basic format method
name = "Bob"
age = 30
print("Name: {}, Age: {}".format(name, age))
# Output: Name: Bob, Age: 30
```

7 Ordering by Position (Name or Number)

The format method supports positional and named arguments for flexible ordering.

```
# Positional ordering
print("{1}, {0}".format("first", "second"))
# Output: second, first

# Named arguments
print("Name: {name}, Age: {age}".format(name="Bob", age=30))
# Output: Name: Bob, Age: 30
```

8 "Repr" Versus String Conversion

The repr specifier (!r) in the format method uses the object's __repr__ method, while the default uses __str__.

```
# Example: Repr vs String
class Person:
    def __str__(self): return "Bob"
    def __repr__(self): return "Person('Bob')"

p = Person()
print("{!r} vs {}".format(p, p))
# Output: Person('Bob') vs Bob
```

9 The spec Field of the format Function and Method

The format specification (inside {}) controls aspects like width, alignment, and precision. The syntax is [fill][align][sign][0][width][,][.precision][type].

9.1 Print-Field Width

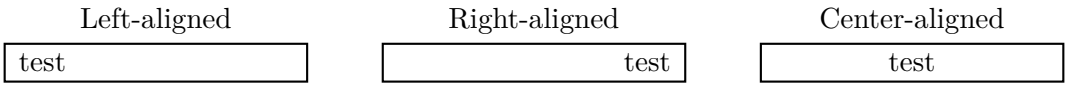
The width specifies the minimum field size.

```
# Example: Field width
print("{:10}".format("test"))
# Output: test
```

9.2 Text Justification: "fill" and "align" Characters

Alignment options are < (left), > (right), $(center), with optional fill characters.$

```
# Example: Alignment
print("{:*^10}".format("test"))
# Output: ***test***
```



9.3 The "sign" Character

The sign (+, -, or space) controls how numbers are displayed.

```
# Example: Sign
print("{:+} vs {: }".format(42, 42))
# Output: +42 vs  42
```

9.4 The Leading-Zero Character (0)

The 0 flag pads numbers with zeros.

```
# Example: Leading zeros
print("{:05d}".format(42))
# Output: 00042
```

9.5 Thousands Place Separator

A comma (,) adds thousands separators.

```
# Example: Thousands separator
print("{:,}".format(1234567))
# Output: 1,234,567
```

9.6 Controlling Precision

The .precision field limits decimal places for floats.

```
# Example: Precision
print("{:.2f}".format(3.14159))
# Output: 3.14
```

9.7 "Precision" Used with Strings (Truncation)

For strings, precision truncates the output.

```
# Example: String truncation
print("{:.3}".format("hello"))
# Output: hel
```

9.8 "Type" Specifiers

Type specifiers like d, f, x, b control the number format.

```
# Example: Type specifiers
print("{:d} {:x} {:b}".format(42, 42, 42))
# Output: 42 2a 101010
```

9.9 Displaying in Binary Radix

The b type displays numbers in binary.

```
# Example: Binary
print("{:b}".format(42))
# Output: 101010
```

9.10 Displaying in Octal and Hex Radix

The o and x/X types display numbers in octal and hexadecimal.

```
# Example: Octal and Hex
print("{:o} {:x} {:X}".format(42, 42, 42))
# Output: 52 2a 2A
```

9.11 Displaying Percentages

The % type converts a number to a percentage.

```
# Example: Percentage
print("{:%}".format(0.75))
# Output: 75.000000%
```

9.12 Binary Radix Example

A practical example of binary formatting:

```
# Example: Binary formatting with width
print("{:08b}".format(42))
# Output: 00101010
```

10 Variable-Size Fields

Variable-size fields use an asterisk (*) to specify width or precision dynamically.

```
# Example: Variable width
width = 10
print("{:*<{width}}".format("test", width=width))
# Output: test******
```

11 Summary

This tutorial covered Python's text formatting techniques, from the percent sign operator to the modern format method. These tools provide precise control over string output, supporting various applications like data reporting and user interfaces.

12 Review Questions

1. What is the difference between %s and %d in the percent sign operator?
2. How does the format method improve upon the % operator?
3. Explain the role of the sign character in formatting numbers.
4. How can you truncate a string using the format method?
5. What is the purpose of the , specifier in number formatting?

13 Suggested Problems

1. Write a program that formats a list of numbers in decimal, hex, and binary with consistent field widths.
2. Create a table of values using the format method with left, right, and center alignment.
3. Format a floating-point number as a percentage with two decimal places.