5. Inserting variables into strings

1. The `format()` Method

The `format()` method is a powerful and flexible way to insert variables into strings. It allows you to replace placeholders in a string with values from variables or expressions. Here's how it works:

```
name = "Alice"
age = 25

# Using positional arguments
print("My name is {} and I'm {} years old.".format(name, age))
# Output: My name is Alice and I'm 25 years old.

# Using keyword arguments
print("My name is {name} and I'm {age} years old.".format(name=name,
age=age))
# Output: My name is Alice and I'm 25 years old.
```

In the examples above, we use the `format()` method to replace the placeholders `{}` with the values of the `name` and `age` variables. The placeholders can be either positional (ordered by position) or keyword-based (using the variable names).

You can also perform formatting operations within the placeholders. For example:

```
pi = 3.14159
print("The value of pi is approximately {:.2f}".format(pi))
# Output: The value of pi is approximately 3.14
```

In this case, the `:.2f` specifies that the value should be formatted as a floating-point number with two decimal places.

2. F-Strings (Python 3.6+)

Starting from Python 3.6, a new string formatting syntax was introduced called f-strings (formatted string literals). F-strings provide a more concise and readable way to insert variables into strings. Here's how they work:

```
name = "Alice"
age = 25

# Using f-strings
print(f"My name is {name} and I'm {age} years old.")
# Output: My name is Alice and I'm 25 years old.
```

In the example above, we use the `f` prefix before the string, and then we can insert variables directly within the string by enclosing them in curly braces `{}`.

F-strings also support formatting operations and expressions:

```
pi = 3.14159
print(f"The value of pi is approximately {pi:.2f}")
# Output: The value of pi is approximately 3.14

radius = 5
print(f"The area of a circle with radius {radius} is {3.14 * radius ** 2:.2f}")
# Output: The area of a circle with radius 5 is 78.54
```

3. The `%` Operator (Older Way)

Before the introduction of the `format()` method and f-strings, Python used the `%` operator for string formatting. While this method is still supported, it is considered less readable and more limited than the newer methods. However, it's still important to understand it, especially when working with legacy code.

```
name = "Alice"
age = 25

# Using the % operator
print("My name is %s and I'm %d years old." % (name, age))
# Output: My name is Alice and I'm 25 years old.
```

In the example above, the `%s` and `%d` placeholders represent a string and an integer, respectively. The values to be inserted are provided as a tuple after the `%` operator.

While the `%` operator is still functional, it is generally recommended to use the `format()` method or f-strings for string formatting in modern Python code, as they provide more flexibility and readability.

See Also

6. Nice formatting Numbers