String

Kittikun Jitpairod

Introduction

In Python, a string is an immutable sequence of characters. It can represent text of any length, from a single character to entire paragraphs.

Creating Strings

Strings can be created using single, double, or triple quotes:

```
single_quoted = 'Hello, World!'
double_quoted = "Python Programming"
triple_quoted = '''This string spans
multiple lines'''
print(single_quoted)
print(double_quoted)
print(triple_quoted)
```

```
Hello, World!
Python Programming
This string spans
multiple lines
```

One quick tip: if you need to include quotes within your string, you can either use different quotes to enclose the string, or you can **escape the quotes**.

```
print('This\'s a book')
```

```
This's a book
```

Common Escape Characters

1. `\n` - Newline
2. `\t` - Tab
3. `\\` - Backslash
4. `\'` - Single quote
5. `\"` - Double quote
6. `\r` - Carriage return
7. `\b` - Backspace
8. `\f` - Form feed

```python
print("Hello\nWorld\n")  # Newline
print("Name:\tJohn")   # Tab
print("Path: C:\\Users\\John")  # Backslash
print('It\'s a beautiful day')  # Single quote in single-quoted string
print("She said, \"Hello!\"")   # Double quote in double-quoted string
```

```
Hello
World

Name:	John
Path: C:\Users\John
It's a beautiful day
She said, "Hello!"
```

String Operations

Concatenation

```python
full_name = "John" + " " + "Doe"
print(full_name)
```

```
John Doe
```

Repetition

```python
repeated = "Python " * 3
print(repeated)
```

```
Python Python Python
```

Indexing and Slicing

```python
text = "Python"
'''
               "P" , "y" , "t" , "h" , "o", "n"
Positive Index  0  ,  1  ,  2  ,  3  ,  4 ,  5
Negative Index -6  , -5  , -4  , -3  , -2 , -1
'''
#### Indexing ####
print(text[0])  # First character
print(text[-1])  # Last character
print(text[3])  # Fourth character from start

#### Slicing: The syntax is [ start : end ] ####
print(text[1:4])  # Characters from index 1 to  3
print(text[:3])  # Characters from the start to index 2
print(text[2:])  # Characters from index 2 to the end
```

```
P
n
h
yth
Pyt
thon
```

Case Conversion

```python
text = "Python Programming"
print(text.upper())
print(text.lower())
print(text.title())
```

```
PYTHON PROGRAMMING
python programming
Python Programming
```

Strip Methods

Remove spaces at the beginning and at the end of the string

```python
text = "   Python   "
print("|" + text.strip() + "|" + "|" + text.lstrip() + "|" + "|" + text.rstrip() + "|")
```

```
|Python||Python   ||   Python|
```

Find and Replace

```python
text = "Python is amazing"
print(text.find("is"))
print(text.replace("amazing", "awesome"))
```

```
7
Python is awesome
```

Split and Join

```python
text = "Python,Java,C++"
languages = text.split(",")
print(languages)

joined = "-".join(languages)
print(joined)
```

```
['Python', 'Java', 'C++']
Python-Java-C++
```

String Formatting

Format Method

```python
name, age = "Bob", 25
print("My name is {} and I am {} years old.".format(name, age))
print("My name is {} and I am {:.2f} years old.".format(name, age))
```

```
My name is Bob and I am 25 years old.
My name is Bob and I am 25.00 years old.
```

% Operator (older style)

```
name, age = "Charlie", 35
print("My name is %s and I am %d years old." %(name, age))
print("My name is %s and I am %.2f years old." %(name, age))
```

```
My name is Charlie and I am 35 years old.
My name is Charlie and I am 35.00 years old.
```

f-strings (Formatted Strings)

F-strings provide a concise and readable way to embed expressions inside string literals. They are prefixed with ‘f’ or ‘F’.

```
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 30 years old.
```

r-strings (Raw Strings)

Raw strings, prefixed with ‘r’ or ‘R’, treat backslashes as literal characters. This is particularly useful for regular expressions and file paths.

```
print(r"C:\Users\John\Documents")
print("C:\\Users\\John\\Documents") # Compare with a normal string:
```

```
C:\Users\John\Documents
C:\Users\John\Documents
```

rf-strings (Raw-Formatted Strings)

RF-strings combine the features of raw strings and formatted strings. They are prefixed with ‘rf’ or ‘RF’.

Syntax and usage:

```
name = "John"
path = r"C:\Users"
print(rf"{path}\{name}")
```

```
C:\Users\John
```

Number Formatting in Strings

formatting options:

- `:.2f`: This specifies we want 2 decimal places for a float.
- `:10`: This sets the total field width to 10 characters.
- `:08.3f`: This pads the number with zeros to a width of 8, including 3 decimal places.
- `:.1%`: This formats the number as a percentage with 1 decimal place.
- `:,`: This adds thousand separators to large numbers.

```python
pi = 3.14159
radius = 5

print(f"Pi to 2 decimal places: {pi:.2f}") # Limiting decimal places
print(f"Pi in a field of 10 characters: {pi:10}") # Specifying width
print(f"Pi padded with zeros: {pi:08.3f}") # Padding with zeros
```

```
Pi to 2 decimal places: 3.14
Pi in a field of 10 characters:    3.14159
Pi padded with zeros: 0003.142
```

```python
area = pi * radius ** 2 # Using expressions
print(f"The area of a circle with radius {radius} is {area:.2f}")

score = 0.8756 # Formatting percentages
print(f"You scored {score:.1%}")

big_number = 1000000 # Formatting large numbers
print(f"A million: {big_number:,}")

scientific = 0.000123321 # Scientific notation
print("Scientific notation: {:.2e}".format(scientific))
```

```
The area of a circle with radius 5 is 78.54
You scored 87.6%
A million: 1,000,000
Scientific notation: 1.23e-04
```