Newton School of Technology

16

Strings

by Gladden Rumao

C01: Fundamentals of Programming

Newton School of Technology

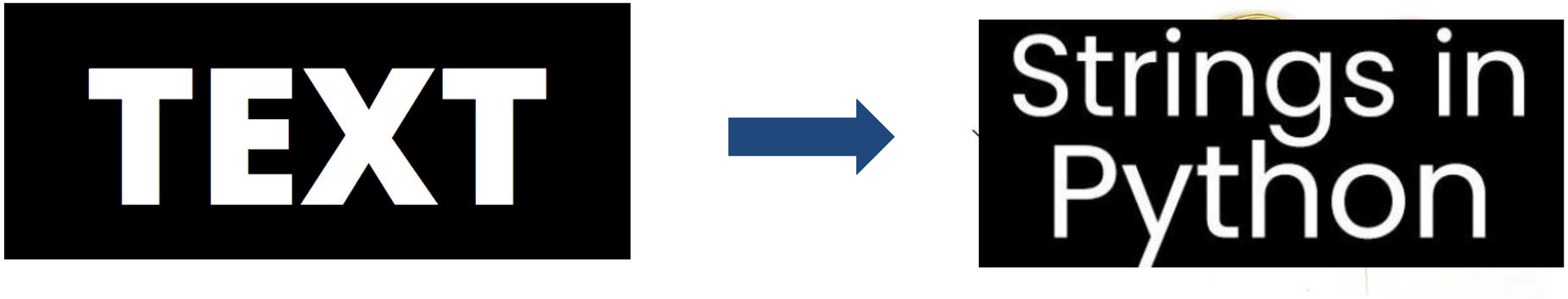
Introduction to Strings

What is a string?

Strings in Python are a fundamental data type **used to represent text**. They are sequences of characters enclosed within single quotes (') or double quotes (").

In Python, there is **no character data type** Python, a character is a string of length one.

When to use strings?

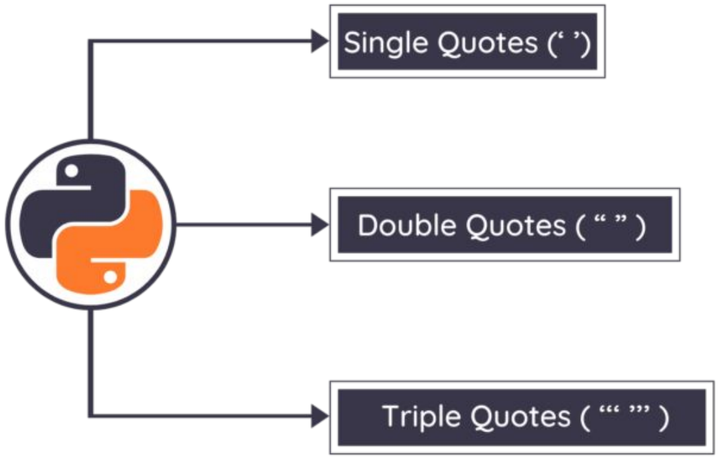


Initializing a string

```python
# Single quotes
single_quote_string = 'Hello, World!'

# Double quotes
double_quote_string = "Hello, World!"

# Triple quotes (for multi-line strings)
multi_line_string = """This is a
multi-line string."""
```



5 star Question

What will be the output ?

```python
print('She said, "Hello!"')
```

Output : She said, “Hello!”

Newton School of Technology

String Indexing

Accessing characters of a string

Strings are indexed, allowing you to access individual characters or substrings using square brackets [].
Indexing starts at 0.

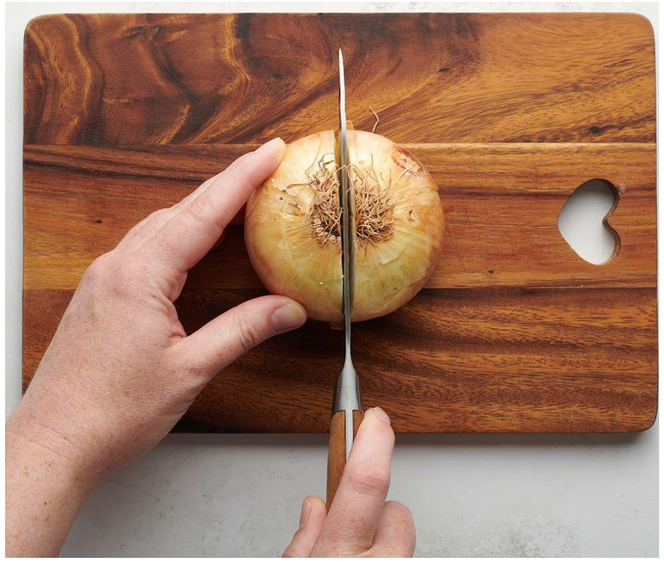
```
example_string = "Python"
first_char = example_string[0]  # 'P'
last_char = example_string[-1]  # 'n'
```

Newton School of Technology

String Slicing

Slicing a string

String slicing is a way to extract a substring from a string using a range of indices. The syntax is **string[start:end]**, where **start** is the index to begin the slice and **end** is the index to end the slice (not inclusive).



Slicing a string: Example

```
# Original string
greeting = "Hello, World!"

# Slicing examples
slice1 = greeting[0:5]
slice2 = greeting[7:12]
slice3 = greeting[-6:-1]
slice4 = greeting[:5]
slice5 = greeting[7:]
```

Slicing a string: Example

```
# Original string
greeting = "Hello, World!"

# Slicing examples
slice1 = greeting[0:5]    # "Hello"
slice2 = greeting[7:12]   # "World"
slice3 = greeting[-6:-1]  # "World"
slice4 = greeting[:5]     # "Hello"
slice5 = greeting[7:]     # "World!"
```

Newton School of Technology

String Methods

String Built-In Methods

- **len(str)**: Returns the length of the string.
- **str.lower()**: Converts the string to lowercase.
- **str.upper()**: Converts the string to uppercase.
- **str.isdigit():** Checks if the string contains only digit or not?
- **str.isalpha():** Checks if the string contains only alphabet or not?
- **str.strip()**: Removes whitespace from both ends.
- **str.replace(a, b)**: Replaces substring **a** with another substring **b** in **str**.
- **str.split('separator')**: Splits the string into a **list** based on a separator.
- **'separator'.join(list)**: Join a **list** of strings into a single string.

String Built-In Methods: Example

```
s = "  Hello World 123  "

print("Length of the string:", len(s))
print("Lowercase:", s.lower())
print("Uppercase:", s.upper())
print("Is the string a digit?", s.isdigit())
print("Is the string alphabetic?", s.isalpha())

stripped_string = s.strip()
print("Stripped string:", stripped_string)

replaced_string = s.replace("World", "Everyone")
print("Replaced string:", replaced_string)
```

String Built-In Methods: Example

```
s = "  Hello World 123  "

print("Length of the string:", len(s))  # 18
print("Lowercase:", s.lower())  # "  hello world 123  "
print("Uppercase:", s.upper())  # "  HELLO WORLD 123  "
print("Is the string a digit?", s.isdigit())  # False
print("Is the string alphabetic?", s.isalpha())  # False

stripped_string = s.strip()
print("Stripped string:", stripped_string)  # "Hello World 123"

replaced_string = s.replace("World", "Everyone")
print("Replaced string:", replaced_string)  # "  Hello Everyone 123  "
```

String Built-In Methods: Split

```
s_default = "apple banana cherry dates"
s_comma = "apple,banana,cherry,dates"


split_default = s_default.split()
print("Split with default delimiter:", split_default)
# ['apple', 'banana', 'cherry', 'dates']

split_comma = s_comma.split(',')
print("Split with ',' delimiter:", split_comma)
# ['apple', 'banana', 'cherry', 'dates']
```

String Built-In Methods: Join

```
1   fruits = ["apple", "banana", "cherry", "dates"]
2   separator = " "
3   all_fruits = separator.join(fruits)
4   print(all_fruits) #apple banana cherry dates
5
6   all_fruits_2 = " ".join(fruits)
7   print(all_fruits_2) #apple banana cherry dates
8
9   all_fruits_3 = ",".join(fruits)
10  print(all_fruits_3) #apple,banana,cherry,dates
```

Unicode representation

- Unicode is a standardized system for encoding text in most of the world's writing systems.
- It assigns a unique number (code point) to each character, regardless of platform, program, or language.

Char	Dec	Binary
!	033	00100001
"	034	00100010
#	035	00100011
$	036	00100100
%	037	00100101
&	038	00100110
'	039	00100111
(	040	00101000
)	041	00101001
*	042	00101010
+	043	00101011
,	044	00101100
-	045	00101101
.	046	00101110
/	047	00101111
0	048	00110000
1	049	00110001
2	050	00110010
3	051	00110011
4	052	00110100
5	053	00110101
6	054	00110110
7	055	00110111

Other Useful Methods

- **ord(ch):** Convert a single Unicode character into its integer representation
- **chr(unicode):** Returns the character that represents the specified unicode.

0 to 9 -> 48 to 57

A to Z -> 65 to 90

a to z -> 97 to 123

Note : Following method takes character/integer as an input and not strings.

Other Useful Methods: Example

```
s = "hello"

# Using ord with a string
print(ord(s[0]))  # Unicode code point of 'h': 104

# Using chr with a Unicode code point
code_point = ord(s[0])
print(chr(code_point))  # Character for code point 104: 'h'
```

Newton School of Technology

String Iteration

String Iteration

For loop

```
text = "Hello, World!"
for char in text:
    print(char)
```

While loop

```
text = "Hello, World!"
index = 0
while index < len(text):
    print(text[index])
    index += 1
```

String Iteration: List Comprehension

When the need is to process or transform each character in a string, list comprehension can be used.

```
text = "Hello, World!"
uppercase_chars = [char.upper() for char in text]
print(uppercase_chars)
```

Output:

```
['H', 'E', 'L', 'L', 'O', ',', ' ', 'W', 'O', 'R', 'L', 'D', '!']
```

Newton School of Technology

String Comparison

String Comparision

String comparision can be done using comparison operators (==, !=, <, >, <=, >=) to check for equality, inequality, and lexicographical order.

Comparison Operators

Operator	Meaning
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

String Comparison: Example

```
str1 = "Hello"
str2 = "World"

# Equality
print(str1 == str2)  # Output: False

# Inequality
print(str1 != str2)  # Output: True

# Greater than
print(str1 > str2)   # Output: False

# Less than
print(str1 < str2)   # Output: True
```

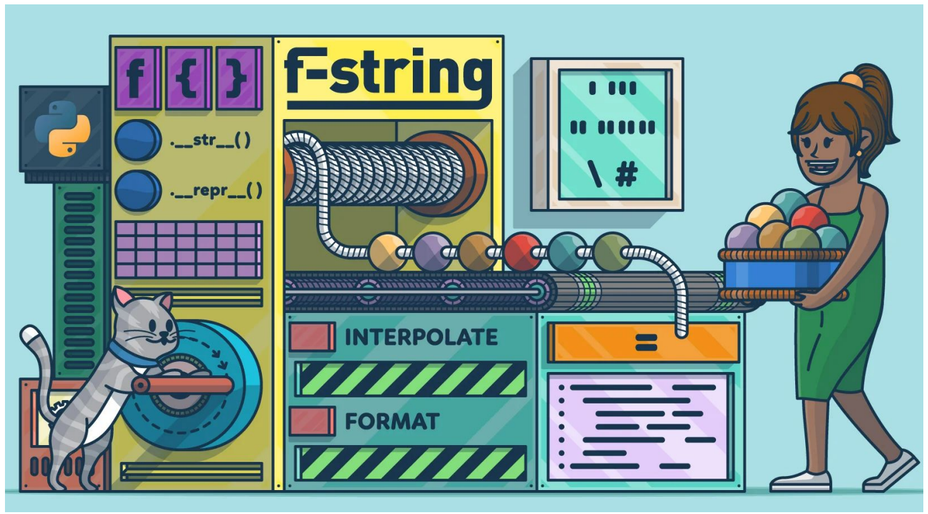
Newton School of Technology

String Formatting

String Formatting

String formatting in Python is a way to construct new strings by embedding values, variables, or expressions within a string template.

3 ways to do it:
1. Old Style (% Operator)
2. New Style (str.format())
3. F-Strings (Python 3.6+)



String Formatting: Example

```
name = "Alice"
age = 30

# Old Style (% Operator)
old_style = 'Name: %s, Age: %d' % (name, age)
print(old_style)

# New Style (str.format())
new_style = 'Name: {}, Age: {}'.format(name, age)
print(new_style)

# F-Strings (Python 3.6+)
f_string = f'Name: {name}, Age: {age}'
print(f_string)
```

Output:

Name: Alice, Age: 30
Name: Alice, Age: 30
Name: Alice, Age: 30

Newton School of Technology

String Concatenation and Repetition

String Concatenation

Strings can be concatenated (joined together) using the **+** operator.

```
# Define individual strings
player = "Manu Bhaker"
achievement = "Olympic Medal Winner"
punctuation = "!!"

# Concatenate the strings
output = player + " is an " + achievement + punctuation

# Print the concatenated string
print(output)

# Output: Manu Bhaker is an Olympic Medal Winner!!
```

String Repetition

Using ***** operator (multiplication operator) a string can be repeated n number of times.

```
word = "Repeat"
print(word * 4)  # Output: RepeatRepeatRepeatRepeat
```

Newton School of Technology

String Immutability

String Immutability: Definition

In Python, strings are immutable, which means that **once a string is created, it cannot be changed**. Any operation that modifies a string will actually create a new string rather than modifying the original one.

String Immutability: Implications

- Directly Modifying a string will give type error.
  - You can use slicing and concatenation to solve the issue.
- Repeated concatenation using ‘+’ operator can cause performance issue due to creation of many intermediate strings.
  - **Solution**: Use str.join()

String Immutability: Explained through code

```python
original = "Immutable"
print(f"Original string ID: {id(original)}")

# Attempting to modify the string
modified = original.replace("Im", "Immutably Im")
print(f"Modified string: {modified}")
print(f"Modified string ID: {id(modified)}")

# Checking if the original string has changed
print(f"Original string remains unchanged: {original}")
print(f"Original string ID remains the same: {id(original)}")
```

```
Original string ID: 140635849403888
Modified string: Immutably Immutable
Modified string ID: 140635849404112
Original string remains unchanged: Immutable
Original string ID remains the same: 140635849403888
```

Newton School of Technology

Escape Characters

Escape Characters

Escape characters in Python are used to insert characters that are illegal in a string. They are preceded by a backslash (\) to indicate that the following character should be interpreted in a special way.
They allow us to include special characters that would otherwise be difficult to represent.

Escape Characters: Example

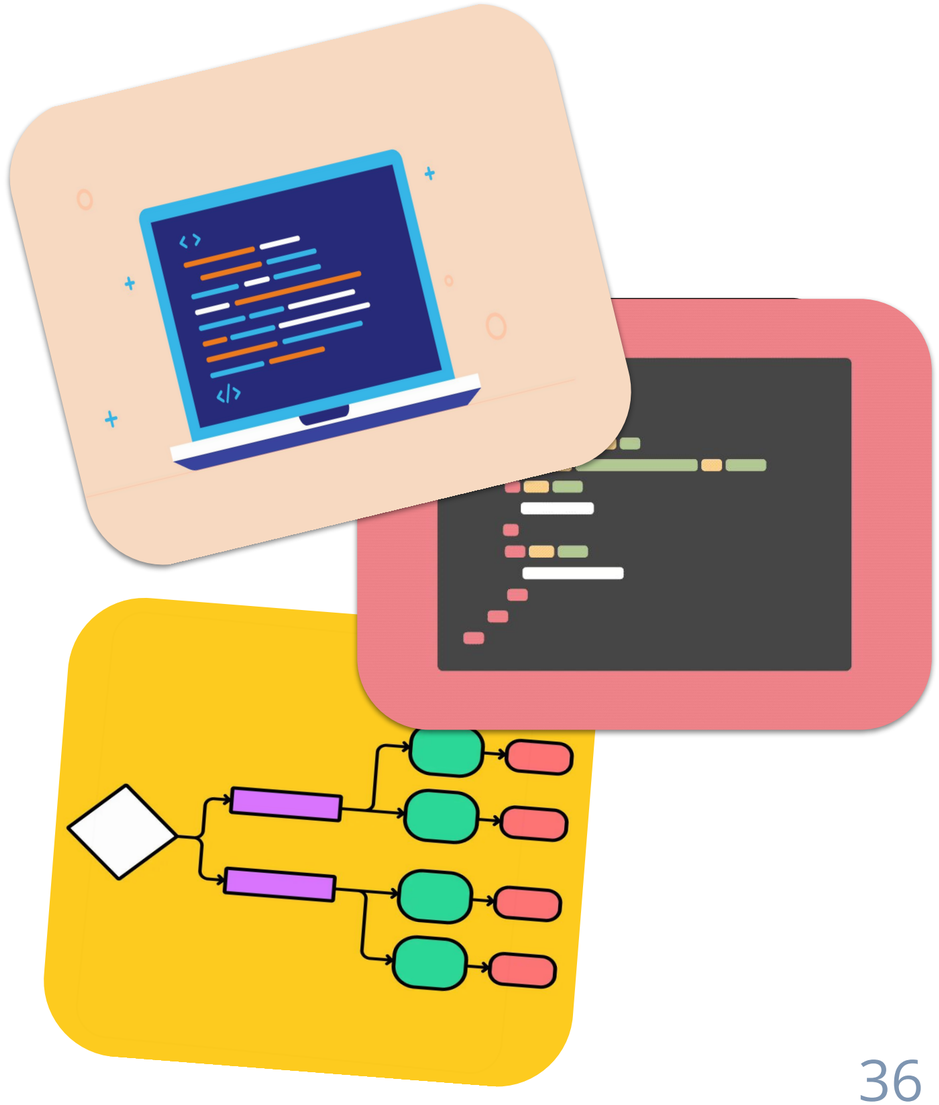
```
text = "Here are some escape characters:\n" \
       "Newline (\\n): This is on a new line.\n" \
       "Tab (\\t):\tThis is tabbed.\n" \
       "Backslash (\\\\): This is a backslash (\\).\n" \
       "Single Quote (\\\'): It\'s a single quote.\n" \
       "Double Quote (\\\"): She said, \"Hello!\"\n"

print(text)
```

```
Here are some escape characters:
Newline (\n): This is on a new line.
Tab (\t):     This is tabbed.
Backslash (\\): This is a backslash (\).
Single Quote (\'): It's a single quote.
Double Quote (\"): She said, "Hello!"
```

Summary

- **When to use strings?**
- **String Indexing and Slicing**
- **String Methods**
  - len(str), str.lower(), str.upper(), str.strip(), str.isdigit(), str.isalpha(), str.replace(a, b)
  - str.split(‘separator), ‘separator’.join(list)
  - ord & chr
- **String Iteration & Comparison**
- **String Concatenation**
- **String Immutability**
- **String Formatting**
- **Escape Characters**



Thank You!