Python for Data Science, AI & Development

Module 1: Python Basics

Table of Contents

Strings Operation in Python

What is a String?

A **string** is a sequence of characters enclosed within ***single quotes (')***, ***double quotes (")***, or ***triple quotes (''' or """`)*** in Python. Strings are one of the most commonly used data types, and they are ***immutable***, meaning their content cannot be changed after they are created.

Basics String Operations

1. Indexing:

Each character in a string has an index starting from 0 (left to right) or -1 (right to left).

Name= "Michael Jackson"

M	i	c	h	a	e	l		J	a	c	k	s	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Name= "Michael Jackson"

M	i	c	h	a	e	l		J	a	c	k	s	o	n
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```python
# Print the element on index 6 in the string
name = "Michael Jackson"
print(name[6])
```

```
l
```

```python
# Print the first element in the string
name = "Michael Jackson"
print(name[-15])
```

```
M
```

2. Slicing:

Extract a part of the string using [start: end] or [start: end: step].

Name= "Michael Jackson"

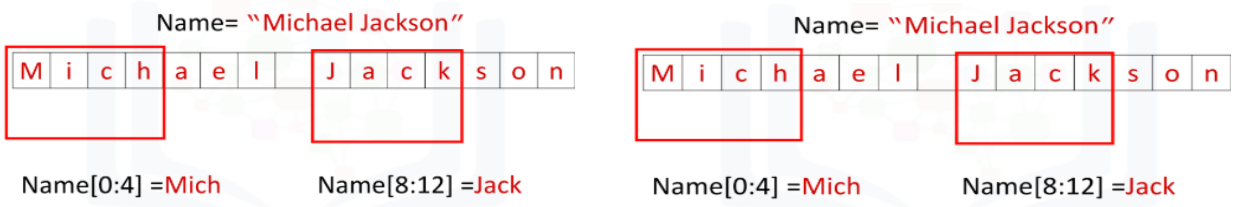
M	i	c	h	a	e	l		J	a	c	k	s	o	n

Name[0:4] =Mich

Name[8:12] =Jack

3. Stride(step)

We can also input a stride value as follows, with the '2' indicating that we are selecting every second variable:




```python
word = "Python"
print(word[0:3])  # Output: Pyt (characters from index 0 to 2)
print(word[::2])  # Output: Pto (every second character)
print(word[::-1])  # Output: nohtyP (reversed string)
```

word[::2] means:

- Start from the beginning (start is not specified, so it defaults to 0).
- Go up to the end of the string (end is not specified, so it defaults to the length of the string).
- Take every **second character** (step is 2).

word[::-1] means:

- ***Start from the end of the string (negative step starts at the last character).***
- Go backwards to the beginning.
- Take **every character** in reverse order (step is -1).

Applicable Example: Check if a String is a Palindrome

```python
text = "madam"
is_palindrome = text == text[::-1]
print(is_palindrome)  # Output: True
```

4. Concatenation

```python
first_name = "John"
last_name = "Doe"

full_name = first_name + " " + last_name
print(full_name)  # Output: John Doe
```

5. Repeating Strings

```python
message = "Hello! "
repeated_message = message * 3
print(repeated_message)  # Output: Hello! Hello! Hello!
```

6. String Length

```python
name = "Python"
print(len(name))  # Output: 6 (number of characters in "Python")
```

Note:

- The len() function in Python is a built-in function used to determine the **length** of an object. Specifically, it returns the number of items in an object such as ***a string, list, tuple, dictionary, or other iterable.***
- ***Object of type 'int' has no len()***

String Methods

1. Changing Case

- upper(): Converts all characters to uppercase.
- lower(): Converts all characters to lowercase.
- capitalize(): Capitalizes the first character.
- title(): Capitalizes the first letter of each word.

```python
text = "hello world"
print(text.upper())        # Output: HELLO WORLD
print(text.lower())        # Output: hello world
print(text.capitalize())   # Output: Hello world
print(text.title())        # Output: Hello World
```

2. Stripping Whitespaces

- strip(): Removes leading and trailing spaces.
- lstrip(): Removes spaces from the left.
- rstrip(): Removes spaces from the right.

```python
text = "   Hello, World!   "
print(text.strip())   # Output: Hello, World!
print(text.lstrip())  # Output: Hello, World!
print(text.rstrip())  # Output:    Hello, World!
```

3. Finding Substrings

- find(): Returns the index of the first occurrence of a substring. Returns -1 if not found.
- index(): Same as find(), but raises an error if the substring is not found.

```python
text = "Hello, World!"
print(text.find("World"))  # Output: 7
print(text.find("Python")) # Output: -1
```

4. Replacing Text

- replace(): Replaces all occurrences of a substring with another string.

```python
text = "I love Python"
new_text = text.replace("Python", "coding")
print(new_text)  # Output: I love coding
```

1. **Splitting and Joining Strings**

   split(): Splits a string into a list based on a delimiter.
   join(): Joins a list of strings into a single string using a delimiter.

```python
text = "apple,banana,cherry"
fruits = text.split(",")  # Splits by comma
print(fruits)             # Output: ['apple', 'banana', 'cherry']

joined_fruits = " - ".join(fruits)  # Joins with " - "
print(joined_fruits)               # Output: apple - banana - cherry
```

6. Checking Content

- startswith(): Checks if a string starts with a given prefix.
- endswith(): Checks if a string ends with a given suffix.
- isalpha(): Checks if all characters are alphabetic.
- isdigit(): Checks if all characters are numeric.
- isalnum(): Checks if all characters are alphanumeric.

```python
text = "Hello123"
print(text.startswith("Hello"))  # Output: True
print(text.endswith("123"))      # Output: True
print(text.isalpha())            # Output: False (contains numbers)
print(text.isdigit())            # Output: False
print(text.isalnum())            # Output: True
```

String Formatting

Python provides multiple ways to format strings for display.

1. **Using f-strings (Python 3.6+)**

   Embed expressions inside curly braces {}

```python
name = "John"
age = 25
print(f"My name is {name}, and I am {age} years old.")
# Output: My name is John, and I am 25 years old.
```

2. Using .format()

```
name = "John"
age = 25
print("My name is {}, and I am {} years old.".format(name, age))
# Output: My name is John, and I am 25 years old.
```

3. Using % Operator (Old Style)

```
name = "John"
age = 25
print("My name is %s, and I am %d years old." % (name, age))
# Output: My name is John, and I am 25 years old.
```

Note:

1. {} is a place holder of string. And. format**(name)**: Replaces the placeholder {} with the value of the variable name.
2. **%s**: Acts as a placeholder for a string value. And **% name**: The value of the variable name is substituted for %s

Escape Sequences

Back slashes represent the beginning of escape sequences. Escape sequences represent strings that may be difficult to input. For example, back slash "n" represents a new line. The output is given by a new line after the back slash "n" is encountered:

Escape special characters using a backslash (\).

- \n: Newline
- \t: Tab
- \\: Backslash is used to place a back slash

```
print("This is a backslash: \\")
# Output: This is a backslash: \
```

- We can also place an “r” before the string to display back slash.

```
print("This is a single backslash: \\")
# Output: This is a single backslash: \
```

- \': Single quote is used to include a single quote inside a string enclosed in double quotes

```
print("It's Python!")  # Output: It's Python!
```

- \": Double quote is used to include **double quotes** inside a string that is enclosed in **double quotes**.

```
print("She said, \"Python is amazing!\"")
# Output: She said, "Python is amazing!"
```

Regular Expressions (RegEx) in Python

What is RegEx?

Regular Expressions (RegEx) are a powerful tool for searching, matching, and manipulating text patterns. In Python, the ***re*** module is used to work with RegEx.

RegEx is like a search query for text patterns. Instead of searching for exact text, it allows you to define patterns that match a wide variety of strings. This RegEx module provides several functions for working with regular expressions, including search, split, findall, and sub.

Key Components of RegEx

1. **Literal Characters**: Match exactly as written (e.g., `a`, `1`, `?`).
2. **Special Characters/Meta-Characters**: Characters with special meanings:
   - `.`: Any single character (except newline).
   - `^`: Start of a string.
   - `$`: End of a string.
   - `*`, `+`, `?`: Quantifiers.
   - `[]`: Match a set of characters.
   - `{}`: Specify the number of occurrences.
   - `|`: Logical OR.
   - `\`: Escape character.
3. **Character Classes**: Predefined character sets.
   - `\d`: Matches digits (0–9).
   - `\w`: Matches word characters (letters, digits, underscore).
   - `\s`: Matches whitespace (spaces, tabs, etc.).

Basic Operations in Python RegEx

1. Importing the re Module

You need to import the re module to use RegEx: *import* ***re***

2. Commonly Used Functions in re

- **re.search()** : Searches for the first match of the pattern in the string.

```
import re

text = "The price is $100."
match = re.search(r"\$\d+", text)  # Search for "$" followed by digits
print(match.group())  # Output: $100
```

r"\$\d+" matches a dollar sign ($) followed by one or more digits.

- **re.match()** : Checks if the pattern matches the **beginning** of the string.

```
import re

text = "Hello, world!"
match = re.match(r"Hello", text)  # Matches only at the start
if match:
    print("Match found!")  # Output: Match found!
```

- **re.findall() :** Finds all occurrences of the pattern in the string and returns them as a list.

```
import re

text = "Email addresses: john@example.com, jane@example.net"
emails = re.findall(r"\b\w+@\w+\.\w+\b", text)  # Find all email addresses
print(emails)  # Output: ['john@example.com', 'jane@example.net']
```

\b Matches a **word boundary** (start or end of a word).
\w+ Matches one or more-word **characters** (letters, digits, or underscore)
@ Matches the literal @ character, which separates the username and domain in an email.
\. Matches a literal dot (.) in the domain.
\b ensures match ends at a **word boundary**.

- **re.split() :** Splits a string by the occurrences of the pattern.

```
import re

text = "apple,banana;cherry orange"
fruits = re.split(r"[,; ]", text)  # Split by comma, semicolon, or space
print(fruits)  # Output: ['apple', 'banana', 'cherry', 'orange']
```

- **re.sub() :** Replaces occurrences of the pattern with a specified string.

```
import re

text = "I love cats. Cats are great!"
result = re.sub(r"cats", "dogs", text, flags=re.IGNORECASE)
print(result)  # Output: I love dogs. Dogs are great!
```

**flags are used in RegX generally.*

- **Escaping Metacharacters**

If you want to match metacharacters as literals, escape them with `\`.

python Copy code

```
pattern = r"\."
string = "This is a test."
print(re.findall(pattern, string))  # Output: ['.']
```

Match Object

When a match is found, a match object is returned, which provides detailed information.

- group(): Returns the matched string.
- start(): Start index of the match.
- end(): End index of the match.
- span(): Tuple of (start, end).

```
pattern = r"world"
string = "hello world"
match = re.search(pattern, string)
if match:
    print("Matched text:", match.group())  # Output: world
    print("Start index:", match.start())  # Output: 6
    print("End index:", match.end())      # Output: 11
    print("Span:", match.span()          # Output: (6, 11)
```

3. Applications of Regex in Python

1. Data Validation: Validating email, phone numbers, and passwords.
2. Text Search: Finding keywords in text files or logs.
3. Text Cleaning: Removing unwanted characters or patterns.

4. Web Scraping: Extracting specific patterns from HTML.
5. Data Extraction: Parsing structured or semi-structured data.