UNIT-2

Strings and Regular Expressions

What is a String?

A string in Python is a sequence of characters enclosed in quotes. It is one of the most commonly used data types for handling text.

Strings and their operations:

1. Concatenation
2. Repetition
3. Slicing
4. Immutability

---

1. Concatenation:

Combining two or more strings using the + operator.

```
str1 = "Hello"
str2 = "World"
result = str1 + " " + str2
print(result)                    # Output: Hello World
```

---

2. Repetition

Repeating a string multiple times using the * operator.

```
greet = "Hi! "
repeated = greet * 3
print(repeated)                    # Output: Hi! Hi! Hi!
```

---

3. Slicing:

Accessing a part of a string using index ranges.

Example:

```
text = "Python Programming"
print(text[0:6])                   # Output: Python
print(text[7:])                    # Output: Programming
print(text[-1])                    # Output: g
print(text[::-1])                  # Output: gnimmargorP nohtyP (reversed string)
```

- text[start:end] → includes start, excludes end.
- text[::-1] → reverses the string.

---

4. Immutability:

Strings are immutable, meaning they cannot be changed after creation.

```
name = "Alice"
# name[0] = 'M'  # ❌ This will raise an error
# To change, you must create a new string
```

```
name = "M" + name[1:]
print(name)
# Output: Mlice
```

Operation	Syntax Example	Description
Concatenation	"A" + "B"	Joins strings
Repetition	"A" * 3	Repeats the string
Slicing	"Python"[0:3]	Extracts a substring
Immutability	Can't change string content	Must create a new string

***String Built-in Methods and Functions:**

Python provides many built-in methods for strings.

Following are the Common String Methods.

Method	Description	Example
.lower()	Converts to lowercase	"HELLO".lower() → 'hello'
.upper()	Converts to uppercase	"hello".upper() → 'HELLO'
.title()	Capitalizes the first letter of each word	"hello world".title() → 'Hello World'
.strip()	Removes leading/trailing spaces	" hello ".strip() → 'hello'
.replace(a, b)	Replaces substring a with b	"cat".replace("c", "b") → 'bat'
.find(sub)	Finds the first index of sub	"apple".find("p") → 1

Method	Description	Example
.count(sub)	Counts occurrences of sub	"banana".count("a") → 3
.startswith(x)	Checks if string starts with x	"data".startswith("d") → True
.endswith(x)	Checks if string ends with x	"data".endswith("a") → True
.split(x)	Splits string into a list	"a,b,c".split(",") → ['a', 'b', 'c']
.join(list)	Joins list elements into a string	" ".join(['Hi', 'there']) → 'Hi there'

Built-in Functions That Work on Strings:

Function	Description	Example
len(s)	Returns length of string	len("Python") → 6
str(x)	Converts x to string	str(123) → '123'
type(s)	Returns type of object	type("abc") → <class 'str'>
ord(c)	Returns Unicode of a character	ord('A') → 65
chr(n)	Returns character of Unicode value	chr(65) → 'A'

Examples:

```
text = "  Hello World  "
print(text.strip().lower().replace("world", "Python"))
# Output: hello python
```

```
words = "one,two,three".split(",")
print(words)                #output ['one', 'two', 'three']
joined = "-".join(words)
print(joined)               #output one-two-three
```

---

*What is a regular Expression?

A Regular Expression (regex) in Python is a sequence of characters that defines a search pattern.

Basic Syntax of Regular Expressions

Regular expressions (regex) are patterns used to match character combinations in strings.

1. Literals

- Matches the exact characters.

cat → matches "cat" in "black cat"

2. Metacharacters

Metacharacters have special meanings

Metacharacter	Example	Matches
. (dot)	a.c	"abc", "a7c" (any one character between a and c)
^ (caret)	^Hello	"Hello world" (matches if the string starts with Hello)
$ (dollar)	world$	"Hello world" (matches if the string ends with world)
* (asterisk)	go*gle	"ggle", "gogle", "google", "gooogle" (zero or more os)
+ (plus)	go+gle	"gogle", "google", but **not** "ggle" (one or more os)

Metacharacter	Example	Matches
? (question mark)	colou?r	"color" or "colour" (zero or one u)
[] (character set)	gr[ae]y	"gray" or "grey"
[^] (negated set)	[^0-9]	Matches any non-digit character
\ (escape)	\.	Matches a literal dot ".", not any character
{n}	\d{3}	"123", but not "12" (exactly 3 digits)
{n,}	\d{2,}	"12", "1234" (2 or more digits)
{n,m}	\d{2,4}	"12", "123", "1234" (between 2 and 4 digits)
() (grouping)	(ha)+	"ha", "hahaha" (one or more "ha" repeats)
`	` (OR)	`cat

---

common regular expression functions — search, match, findall, and sub — with explanations and examples in **Python** using the re module:

1. re.search()

◈ Purpose:

Searches the **entire string** for the first match.

◈ Syntax:

re.search(pattern, string)

◈ Example:

import re

text = "My phone number is “123-456-7890"

result = re.search(r'\d{3}-\d{3}-\d{4}', text)

print(result.group())

◈ **Output**: 123-456-7890

---

2. re.match()

◈ **Purpose:**

Checks for a match only at the beginning of the string.

◈ **Syntax:**

re.match(pattern, string)

◈ **Example:**

text = "Hello World"

result = re.match(r'Hello', text)

print(result.group())

◈ **Output**: Hello

**If you try re.match(r'World', text), it returns None because it doesn't start the string.

---

3. re.findall()

◈ **Purpose:**

To extract all email addresses from a block of text.

◈ **Syntax:**

re.findall(pattern, string)

◈ **Example:**

text = "Emails: test1@gmail.com, test2@yahoo.com"

results = re.findall(r'\w+@\w+\.\w+', text)

print(results)

◈ **Output**: ['test1@gmail.com', 'test2@yahoo.com']

---

4. re.sub()

🔹 **Purpose:**

Substitutes matches in the string with a replacement.

🔹 **Syntax:**

re.sub(pattern, replacement, string)

🔹 **Example:**

text = "The price is $100"

result = re.sub(r'\$\d+', '$200', text)

print(result)

🔹 **Output**: The price is $200

Summary Table

Function	Description	Returns
search()	First match anywhere in string	Match object or None
match()	Match only at start of string	Match object or None
findall()	All matches in list form	List of strings
sub()	Replace match with new string	Modified string

Sequence: Lists

1. Introduction to Lists

- A **list** is a mutable, ordered collection of items in Python.
- Lists can contain elements of any data type, including numbers, strings, other lists, or mixed types.
- Lists are defined using square brackets [].

Example

fruits = ['apple', 'banana', 'cherry']

---

2. Operations on Lists

a. Accessing Elements

- Elements are accessed using zero-based indexing.

```
# Access the first and third items
print(fruits[0])          # Output: apple
print(fruits[2])          # Output: cherry
```

- Negative indexing can be used to access elements from the end of the list.

```
print(fruits[-1])          # Output: cherry (last item)
```

---

b. Slicing

- Slicing returns a new list containing a subset of the original list.

```
# Syntax: list[start:stop:step]
print(fruits[0:2])    # Output: ['apple', 'banana']
print(fruits[::2])    # Output: ['apple', 'cherry']
```

- Omitting indices uses defaults (start=0, end=len(list)).

---

c. Modifying Elements

- Lists are mutable, so elements can be changed or reassigned.

```
fruits[1] = 'blueberry'
print(fruits)  # Output: ['apple', 'blueberry', 'cherry']
```

- You can also modify multiple elements via slice assignment.

```
fruits[0:2] = ['mango', 'grape']
print(fruits)  # Output: ['mango', 'grape', 'cherry']
```

Sequence: Lists – Methods:

Creation

```
my_list = [1, 2, 3]
```

Add / Insert

Method	Description	Example
append(x)	Adds item x to the end	my_list.append(4)
extend(iterable)	Adds all items from an iterable	my_list.extend([5, 6])
insert(i, x)	Inserts x at index i	my_list.insert(1, 'a')

Remove

Method	Description	Example
remove(x)	Removes first occurrence of x	my_list.remove(2)
pop([i])	Removes and returns item at index i	my_list.pop()
clear()	Removes all elements from the list	my_list.clear()_

Tuple:

A **tuple** is an ordered, immutable (unchangeable) collection of elements. Tuples are defined using **parentheses ()**.

1. Basic Tuple Creation:
Write a Python program to create a tuple and display its elements.

1: Basic Tuple Creation

```
my_tuple = ("apple", "banana", "cherry")
print("Tuple elements:", my_tuple)
```

2. Access Tuple Elements:
Write a Python program to access and print the first and last elements of a tuple.

```
fruits = ("apple", "banana", "mango", "grape")
print("First fruit:", fruits[0])
print("Last fruit:", fruits[-1])
```

---

3. Tuple Slicing

Write a Python program to slice a tuple and display the result.

numbers = (10, 20, 30, 40, 50, 60)

print("Sliced tuple [1:4]:", numbers[1:4])

❖ Sets - Introduction

A **set** is a **collection of distinct elements**. Sets are often used in mathematics and computer science to group related objects.

Set Notation

- A set is usually denoted by **curly braces** {}.
- Example:
  A = {1, 2, 3} is a set of three elements.

Properties of Sets

- **Unordered**: The order of elements doesn't matter.
  {1, 2} is the same as {2, 1}.
- **Unique elements**: No duplicates.
  {1, 2, 2, 3} is just {1, 2, 3}.

❖ Set Operations

1. Union (∪):

The union of two sets is a set containing all elements from both sets, without duplicates.

🔷 ***Notation:***

A ∪ B

🔷 ***Example:***

If A = {1, 2, 3} and B = {3, 4, 5},
then A ∪ B = {1, 2, 3, 4, 5}

2. **Intersection (∩)**

The intersection of two sets is a set containing only the elements common to both sets.

🔷 Notation:

A ∩ B

🔷 Example:

If A = {1, 2, 3} and B = {3, 4, 5},
then A ∩ B = {3}

3. **Difference (-):**

The difference between two sets (A − B) is a set of elements in A but not in B.

◈ Notation:

A − B

◈ Example:

If A = {1, 2, 3} and B = {3, 4, 5},
then A − B = {1, 2}
(because 3 is in both, and we remove it from A)

Python Program:

1,You're building a simple program to:

- **Store students enrolled in different courses.**
- **Show:**
  - **All students (union)**
  - **Students enrolled in both courses (intersection)**
  - **Students in one course but not the other (difference)**
  - **Students enrolled in only one course (symmetric difference)**

```
# Students enrolled in each course
math_students = {"Alice", "Bob", "Charlie", "David"}
science_students = {"Charlie", "David", "Eve", "Frank"}


print("Math students:", math_students)
print("Science students:", science_students)
# Math students: {'Charlie', 'Alice', 'Bob', 'David'}
Science students: {'Eve', 'Charlie', 'Frank', 'David'}
# Union - All unique students
all_students = math_students.union(science_students)
print("\nAll students (Union):", all_students)
# All students (Union): {'Charlie', 'Alice', 'Bob', 'Frank', 'Eve', 'David'}
```

```
# Intersection - Students in both courses
both_courses = math_students.intersection(science_students)
print("Students in both Math and Science (Intersection):", both_courses)
# Students in both Math and Science (Intersection): {'Charlie', 'David'}
# Difference - Only in Math
only_math = math_students.difference(science_students)
print("Students only in Math (Difference):", only_math)
# Students only in Math (Difference): {'Alice', 'Bob'}
# Difference - Only in Science
only_science = science_students.difference(math_students)
print("Students only in Science (Difference):", only_science)
# Students only in science (Difference): {'Eve', frank'}
# Symmetric Difference - Students in only one course
only_one_course = math_students.symmetric_difference(science_students)
print("Students in only one course (Symmetric Difference):", only_one_course)
# Students in only one course (Symmetric Difference): {'Alice', 'Bob', 'Eve', 'Frank'}
```

❖ Frozenset

A frozenset is an immutable version of a set. Once created, it cannot be changed (no add/remove).

◈ Creating a Frozenset:

```
fs = frozenset([1, 2, 3])
```

◈ Allowed operations:

- .union(), .intersection(), .difference()
- You cannot use .add() or .remove() on frozensets.

◆ Example:

```
fs1 = frozenset([1, 2, 3])
```

```
fs2 = frozenset([3, 4])

print(fs1.union(fs2))  # Output: frozenset({1, 2, 3, 4})
```

---

3. Bytes

A bytes object is an immutable sequence of bytes (integers from 0 to 255), used for binary data (e.g. files, networking).

🔹 Creating a bytes object:

```
b = bytes([65, 66, 67])   # ASCII values

print(b)  # Output: b'ABC'
```

- Cannot modify contents: b[0] = 70 → ❌ Error

---

4. Byte array

A byte array is a mutable version of bytes.

🔹 Creating a byte array:

```
ba = bytearray([65, 66, 67])

print(ba)          # Output: bytearray(b'ABC')
```

🔹 Modifying:

```
ba[0] = 70

print(ba)                  # Output: bytearray(b'ABC')
```
