Regular Expressions in Python

Prepared by: Soumya Patil

Introduction to Regular Expressions

- Regular Expressions (Regex) are patterns used to match strings.
- They are powerful tools for searching, extracting, and modifying text.

Without Regex:

- - Manual string searching using loops and conditions.

With Regex:

- - Use `re` module in Python for pattern matching and extraction.

Finding Patterns of Text - Without and With Regex

Without Regex:

```
text = "My phone number is 9876543210"
for word in text.split():
    if word.isdigit() and len(word) == 10:
        print(word)
```

With Regex:

```
import re
text = "My phone number is 9876543210"
match = re.findall(r'\d{10}', text)
print(match)
```

RegEx Functions

- The re module offers a set of functions that allows us to search a string for a match:

Function	Description
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

Metacharacters

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he{2}o"
|	Either or	"falls|stays"
()	Capture and group	

Special Sequences

- A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description	Example
\A	Returns a match if the specified characters are at the beginning of the string	"\AThe"
\b	Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string contains digits (numbers from 0-9)	"\d"
\D	Returns a match where the string DOES NOT contain digits	"\D"
\s	Returns a match where the string contains a white space character	"\s"
\S	Returns a match where the string DOES NOT contain a white space character	"\S"
\w	Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore _ character)	"\w"
\W	Returns a match where the string DOES NOT contain any word characters	"\W"
\Z	Returns a match if the specified characters are at the end of the string	"Spain\Z"

Sets

- A set is a set of characters inside a pair of square brackets [] with a special meaning:

Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) is present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[^arn]	Returns a match for any character EXCEPT a, r, and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

Greedy and Non-Greedy Matching

Greedy: Matches as much text as possible

- Example: re.findall(r'<.*>', '<tag>Text</tag>') → ['<tag>Text</tag>']

Non-Greedy: Matches as little text as possible

- Example: re.findall(r'<.*?>', '<tag>Text</tag>') → ['<tag>', '</tag>']

Character Classes

Common character classes:

\d → digits

\w → alphanumeric characters

\s → whitespace

- Example:

re.findall(r'\d+', 'There are 24 students and 3 teachers.') → ['24', '3']

Review of Common Regex Symbols

- \d - Digit
- \w - Word character
- \s - Whitespace
- . - Any character
- ^ - Start of string
- $ - End of string
- + - One or more
- * - Zero or more
- ? - Optional (or non-greedy)
- [] - Character set

The findall() Method

- findall() returns all non-overlapping matches of a pattern in a string.

Example:

```
import re
text = "My emails are test@example.com and hello@abc.org"
emails = re.findall(r'[\w.-]+@[\w.-]+', text)
print(emails)
```

The findall() Method

Example

Return an empty list if no match was found:

```
import re

txt = "The rain in Spain"
x = re.findall("Portugal", txt)
print(x)
```

Making Your Own Character Classes

Custom classes use square brackets [].

Example:
re.findall(r'[aeiou]', 'Python Programming')
['o', 'o', 'a', 'i']

#Negation:
re.findall(r'[^aeiou]', 'Python')
['P', 'y', 't', 'h', 'n']

The Caret (^) and Dollar ($) Symbols

^ → Matches start of string

$ → Matches end of string

Example:

re.findall(r'^Hello', 'Hello World')

output: ['Hello']

re.findall(r'World$', 'Hello World')

output: ['World']

The search()

- The search() function searches the string for a match, and returns a Match object if there is a match.
- If there is more than one match, only the first occurrence of the match will be returned:

The Search()

- Example
- Search for the first white-space character in the string:

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:“, x.start())
```

The sub() Function

The sub() function replaces the matches with the text of your choice:

Example

Replace every white-space character with the number 9:

```
import re

txt = " Happy Kannada Rajyotsava 2025 "
x = re.sub("\s", "9", txt)
print(x)
```

The sub() Function

You can control the number of replacements by specifying the count parameter:

Example

#Replace the first 2 occurrences:

```
import re

txt = " Happy Kannada Rajyotsava 2025 "
x = re.sub("\s", "9", txt, 2)
print(x)
```

Match Object

A Match Object is an object containing information about the search and the result.

Note: If there is no match, the value None will be returned, instead of the Match Object.

Example

Do a search that will return a Match Object:

```
import re

txt = "Happy Kannada Rajyotsava 2025 "
x = re.search(“Ra", txt)
print(x) #this will print an object
```

Example

Print the string passed into the function:

```
import re

txt = " Happy Kannada Rajyotsava 2025 "
x = re.search(r"\bS\w+", txt)
print(x.string)
```

The Wildcard Character

. → Matches any single character (except newline)

Example:

re.findall(r'c.t', 'cat, cot, cut, coat’)

output

['cat', 'cot', 'cut']

shelve Module

- shelve is a built-in Python module used for persistent storage of Python objects.
- It allows you to save variables (objects) to a file and retrieve them later — like a simple database.
- Works similar to a dictionary, where data is stored as key–value pairs.

Example:

import shelve

Open a shelf file → store Python objects

Close the file → data is automatically saved

Key Features:

- Stores data on disk (persistent storage)
- Keys must be strings; values can be any picklable object
- Easy to use — no need for SQL or manual serialization

Saving and Retrieving Variables

```
import shelve

# Open a shelf file
with shelve.open("mydata") as shelf:
    shelf["name"] = "ABC"
    shelf["age"] = 25
    shelf["marks"] = [85, 90, 92]

# Retrieve data
with shelve.open("mydata") as shelf:
    print(shelf["name"])
    print(shelf["marks"])
```

Advantages, Limitations, and Use Cases

- **Advantages:**
  - ✅ Simple way to save variables
  - ✅ No need for SQL or external libraries
  - ✅ Automatically handles complex data types (lists, dicts, etc.)
- **Limitations:**
  - ⚠️ Only one program should write to a shelf at a time
  - ⚠️ Keys must be strings
  - ⚠️ Not suitable for very large datasets or concurrent access

Common Use Cases:

- Storing user preferences
- Caching computed data
- Saving session variables between program runs

Serialization

- **Serialization** means **converting a Python object into a byte stream** (a sequence of bytes) so it can be:
- **Stored on disk** (in a file), or
- **Sent over a network**, and later
- **Reconstructed (deserialized)** back into the original Python object.

Example:

```
my_data = {"name": "Soumya", "marks": [85, 90, 92]}
```

- Before saving it to a file, Python must **convert** this dictionary into a **storable format** (bytes).
  That process is **serialization**.

How does shelve use pickle

- The **pickle module** in Python is used to **serialize and deserialize** Python objects.
- The **shelve module uses pickle automatically inside** — you don't have to do it yourself.

shelf["marks"] = [85, 90, 92]

- Internally, shelve performs:
- **pickle.dump([85, 90, 92], file)** → converts list to bytes and stores it.
- When you later read it, it does **pickle.load(file)** → converts bytes back to the list.

Term	Meaning
Serialization	Converting Python objects → bytes
Deserialization	Converting bytes → Python objects
pickle	Python's built-in module that does serialization
shelve	Uses pickle internally to save/load Python variables automatically

Practise Programs

Write a Python program using regular expression to extract all email addresses from a given paragraph.

- Write a Python program using regular expression to whether a phone number is valid (formats: 123-456-7890 or (123) 456-7890).

- Write a Python program using regular expression to count letters, digits, and spaces in a sentence using regex.

- Write a Python program to check if a string starts and ends with the same word (use ^ and $).

- Verifies whether each chatbot response starts with a polite greeting such as "Hi", "Hello", or "Dear" (use ^ anchor).