**Regular Expression in Python**

* The **regular expressions** can be defined as the sequence of characters which are used to search for a pattern in a string.
* The module re provides the support to use regex in the python program.
* The **re** module throws an exception if there is some error while using the regular expression.
* The **re** module must be imported to use the regex functionalities in python.

**Syntax:**

**import** re

**Regex Functions**

* The following regex functions are used in the python.

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Function** | **Description** |
| 1 | match | This method matches the regex pattern in the string with the optional flag. It returns true if a match is found in the string otherwise it returns false. |
| 2 | search | This method returns the match object if there is a match found in the string. |
| 3 | findall | It returns a list that contains all the matches of a pattern in the string. |
| 4 | split | Returns a list in which the string has been split in each match. |
| 5 | sub | Replace one or many matches in the string. |

**Forming a regular expression**

* A regular expression can be formed by using the mix of meta-characters, special sequences, and sets.

### Meta-Characters

* Metacharacter is a character with the specified meaning.

|  |  |  |
| --- | --- | --- |
| **Metacharacter** | **Description** | **Example** |
| [] | It represents the set of characters. | "[a-z]" |
| \ | It represents the special sequence. | "\r" |
| . | It signals that any character is present at some specific place. | "Ja.v." |
| ^ | It represents the pattern present at the beginning of the string. | "^Java" |
| $ | It represents the pattern present at the end of the string. | "point" |
| \* | It represents zero or more occurrences of a pattern in the string. | "hello\*" |
| + | It represents one or more occurrences of a pattern in the string. | "hello+" |
| {} | The specified number of occurrences of a pattern the string. | "java{2}" |
| | | It represents either this or that character is present. | "java|point" |
| () | Capture and group |  |

### Special Sequences

* Special sequences are the sequences containing \ followed by one of the characters.

|  |  |
| --- | --- |
| **Character** | **Description** |
| \A | It returns a match if the specified characters are present at the beginning of the string. |
| \b | It returns a match if the specified characters are present at the beginning or the end of the string. |
| \B | It returns a match if the specified characters are present at the beginning of the string but not at the end. |
| \d | It returns a match if the string contains digits [0-9]. |
| \D | It returns a match if the string doesn't contain the digits [0-9]. |
| \s | It returns a match if the string contains any white space character. |
| \S | It returns a match if the string doesn't contain any white space character. |
| \w | It returns a match if the string contains any word characters. |
| \W | It returns a match if the string doesn't contain any word. |
| \Z | Returns a match if the specified characters are at the end of the string. |

### Sets

* A set is a group of characters given inside a pair of square brackets. It represents the special meaning.

|  |  |  |
| --- | --- | --- |
| **S.No** | **Set** | **Description** |
| 1 | [arn] | Returns a match if the string contains any of the specified characters in the set. |
| 2 | [a-n] | Returns a match if the string contains any of the characters between a to n. |
| 3 | [^arn] | Returns a match if the string contains the characters except a, r, and n. |
| 4 | [0123] | Returns a match if the string contains any of the specified digits. |
| 5 | [0-9] | Returns a match if the string contains any digit between 0 and 9. |
| 6 | [0-5][0-9] | Returns a match if the string contains any digit between 00 and 59. |
| 10 | [a-zA-Z] | Returns a match if the string contains any alphabet (lower-case or upper-case). |

### The findall() function

* This method returns a list containing a list of all matches of a pattern within the string.
* It returns the patterns in the order they are found. If there are no matches, then an empty list is returned.

**Example:**

import re

str = "How are you. How is everything"

matches = re.findall("How", str)

print(matches)

print(matches)

**Output:**

['How', 'How']

['How', 'How']

**The match object**

* The match object contains the information about the search and the output.
* If there is no match found, the None object is returned.

**Example:**

import re

str = "How are you. How is everything"

matches = re.search("How", str)

print(type(matches))

print(matches) #matches is the search object

**Output:**

<class '\_sre.SRE\_Match'>

<\_sre.SRE\_Match object; span=(0, 3), match='How'>

### The Match object methods

There are the following methods associated with the Match object.

1. **span():** It returns the tuple containing the starting and end position of the match.
2. **string():** It returns a string passed into the function.
3. **group():** The part of the string is returned where the match is found.

**Example:**

import re

str = "How are you. How is everything"

matches = re.search("How", str)

print(matches.span())

print(matches.group())

print(matches.string)

**Output:**

(0, 3)

How

How are you. How is everything

# **Quantifiers**

* **Quantifiers** are the mechanisms to define how a **character**, **metacharacter**, or **character set** can be **repeated**.

Here is the list of 4 basic quantifers:

| **Symbol** | **Name** | **Quantification of previous character** |
| --- | --- | --- |
| ? | Question Mark | Optional (0 or 1 repetitions) |
| \* | Asterisk | Zero or more times |
| + | Plus Sign | One or more times |
| {n,m} | Curly Braces | Between n and m times |

Let us go through different examples to understand them one by one.

### Example 1

Find all the matches for dog and dogs in the given text.

Input:

**import** **re**

txt = """

I have 2 dogs. One dog is 1 year old and other one is 2 years old. Both dogs are very cute!

"""

pattern = re.compile("dogs?")

pattern.findall(txt)

Output:

['dogs', 'dog', 'dogs']

### Example 2

Find all filenames starting with file and ending with .txt in the given text.

Input:

txt = """

file1.txt

file\_one.txt

file.txt

fil.txt

file.xml

file-1.txt

"""

pattern = re.compile("file[\w-]\*\.txt")

pattern.findall(txt)

Output:

['file1.txt', 'file\_one.txt', 'file.txt', 'file-1.txt']

### Example 3

Find all filenames starting with file followed by 1 or more digits and ending with .txt in the given text.

Input:

txt = """

file1.txt

file\_one.txt

file09.txt

fil.txt

file23.xml

file.txt

"""

pattern = re.compile("file\d+\.txt")

pattern.findall(txt)

Output:

['file1.txt', 'file09.txt']

We can use the curly brackets syntax here with these modifications:

| **Syntax** | **Description** |
| --- | --- |
| {n} | The previous character is repeated exactly n times. |
| {n,} | The previous character is repeated at least n times. |
| {,n} | The previous character is repeated at most n times. |
| {n,m} | The previous character is repeated between n and m times (both inclusive). |

### Example 4

Find years in the given text.

Input:

txt = """

The first season of Indian Premiere League (IPL) was played in 2008.

The second season was played in 2009 in South Africa.

Last season was played in 2018 and won by Chennai Super Kings (CSK).

CSK won the title in 2010 and 2011 as well.

Mumbai Indians (MI) has also won the title 3 times in 2013, 2015 and 2017.

"""

pattern = re.compile("\d**{4}**")

pattern.findall(txt)

Output:

['2008', '2009', '2018', '2010', '2011', '2013', '2015', '2017']

### Example 5

In the given text, filter out all 4 or more digit numbers.

Input:

txt = """

123143

432

5657

4435

54

65111

"""

pattern = re.compile("\d{4,}")

re.findall(pattern, txt)

Output:

['123143', '5657', '4435', '65111']

### Example 6

Write a pattern to validate telephone numbers.

Telephone numbers can be of the form: 555-555-5555, 555 555 5555, 5555555555

Input:

txt = """

555-555-5555

555 555 5555

5555555555

"""

pattern = re.compile("\d**{3}**[-\s]?\d**{3}**[-\s]?\d**{4}**")

pattern.findall(txt)

Output:

['555-555-5555', '555 555 5555', '5555555555']