5.1 REGULAR EXPRESSIONS

Regular expressions are a powerful tool for various kinds of string manipulation. These are basically a special text string that is used for describing a search pattern to extract information from text such as code, files, log, spreadsheets, or even documents.

Regular expressions are a *domain specific language* (DSL) that is present as a library in most of the modern programming languages, besides Python. A *regular expression* is a special sequence of characters that helps to match or find strings in another string. In Python, regular expressions can be accessed using the **re** module which comes as a part of the Standard Library. In this section, we will discuss some important methods in the **re** module.

Programming Tip: An exception re.error is raised if any error occurs while compiling or using regular expressions.

5.1.1 The `match()` Function

As the name suggest, the `match()` function matches a pattern to a string with optional flags. The syntax of `match()` function is,

```
re.match(pattern, string, flags=0)
```

The function tries to match the pattern (which specifies the regular expression to be matched) with a string (that will be searched for the pattern at the beginning of the string). The flag field is optional. Some values of flags are specified in the Table 6.4. To specify more than one flag, you can use the bitwise `OR` operator as in `re.I | re.M.` If the `re.match()` function finds a match, it returns the match object and `None` otherwise.

Table 6.4 Different values of flags

Flag	Description
re.I	Case sensitive matching
re.M	Matches at the end of the line
re.X	Ignores whitespace characters
re.U	Interprets letters according to Unicode character set

Example 6.26 Program to demonstrate the use of `match()` function

```
import re
string = "She sells sea shells on the sea shore"
pattern1 = "sells"
if re.match(pattern1, string):
    print("Match Found")
else:
    print(pattern1, "is not present in the string")
pattern2 = "She"
if re.match(pattern2, string):
    print("Match Found")
else:
    print(pattern2, "is not present in the string")
```

OUTPUT

```
sells is not present in the string
Match Found
```

In the above program, 'sells' is present in the string but still we got the output as match not found. This is because the `re.match()` function finds a match only at the beginning of the string. Since, the word 'sells' is present in the middle of the string, hence the result.

Note On success, match() function returns an object representing the match, else returns `None`.

5.1.2 The `search()` Function

In the previous function, we saw that even when the pattern was present in the string, `None` was returned because the match was done only at the beginning of the string. So, we have another function, i.e. `search()`, in the `re` module that searches for a pattern anywhere in the string. The syntax of the `search()` function can be given as,

Programming Tip: While using regular expressions, always use raw strings.

```
re.search(pattern, string, flags=0)
```

The syntax is similar to the `match()` function. The function searches for first occurrence of *pattern* within a *string* with optional *flags*. If the search is successful, a *match* object is returned and `None` otherwise.

Example 6.1 Program to demonstrate the use of `search()` function

```
import re
string = "She sells sea shells on the sea shore"
pattern = "sells"
if re.search(pattern, string):
    print("Match Found")
else:
    print(pattern, "is not present in the string")
```

OUTPUT

```
Match Found
```

Note The `re.search()` finds a match of a pattern anywhere in the string.

5.1.3 The `sub()` Function

The `sub()` function in the `re` module can be used to search a pattern in the string and replace it with another pattern. The syntax of `sub()` function can be given as,

```
re.sub(pattern, repl, string, max=0)
```

According to the syntax, the `sub()` function replaces all occurrences of the `pattern` in `string` with `repl`, substituting all occurrences unless any `max` value is provided. This method returns a modified string.

Example 6.2 Program to demonstrate the use of `sub()` function

```
import re
string = "She sells sea shells on the sea shore"
pattern = "sea"
repl = "ocean"
new_string = re.sub(pattern, repl, string, 1)
print(new_string)
```

OUTPUT

```
She sells ocean shells on the sea shore
```

In the above program, note that only one occurrence was replaced and not all because we had provided `1` as the value of `max`.

5.1.4 The `findall()` and `finditer()` Functions

The `findall()` function is used to search a string and returns a list of matches of the pattern in the string. If no match is found, then the returned list is empty. The syntax of `match()` function can be given as,

```
matchList = re.findall(pattern, input_str, flags=0)
```

Example 6.3 Program to demonstrate the use of `findall()` function

```
import re
pattern = r"[a-zA-Z]+ \d+"
matches = re.findall(pattern, "LXI 2013, VXI 2015, VDI 20104, Maruti Suzuki Cars in
India")
for match in matches:
    print(match, end = " ")
```

OUTPUT

```
LXI 2013  VXI 2015  VDI 20104
```

Note The `re.findall()` function returns a list of all substrings that match a pattern.

In the above code, the regular expression, `pattern = r"[a-zA-Z]+ \d+"`, finds all patterns that begin with one or more characters followed by a space and then followed by one or more digits.

The `finditer()` function is same as `findall()` function but instead of returning match objects, it returns an iterator. This iterator can be used to print the index of match in the given string.

Example 6.4 Program to demonstrate the use of `finditer()` function

```
import re
pattern = r"[a-zA-Z]+ \d+"
matches = re.finditer(pattern, "LXI 2013, VXI 2015, VDI 20104, Maruti Suzuki Cars
availble with us")
for match in matches:
    print("Match found at starting index : ", match.start())
    print("Match found at ending index : ", match.end())
    print("Match found at starting and ending index : ", match.span())
```

OUTPUT

```
Match found at starting index :  0
Match found at ending index :  8
Match found at starting and ending index :  (0, 8)
Match found at starting index :  10
Match found at ending index :  18
Match found at starting and ending index :  (10, 18)
Match found at starting index :  20
Match found at ending index :  29
Match found at starting and ending index :  (20, 29)
```

Note that the `start()` function returns the starting index of the first match in the given string. Similarly, we have `end()` function which returns the ending index of the first match. Another method, `span()` returns the starting and ending index of the first match as a tuple.

Note The match object returned by `search()`, `match()`, and `findall()` functions have `start()` and `end()` methods, that returns the starting and ending index of the first match.

5.1.5 Flag Options

The `search()`, `findall()`, and `match()` functions of the module take options to modify the behavior of the pattern match. Some of these flags are:

`re.I or re.IGNORECASE`—Ignores case of characters, so `"Match", "MATCH", "mAtCh",` etc are all same

`re.S or re.DOTALL`—Enables dot (`.`) to match newline character. By default, dot matches any character other than the newline character.

`re.M or re.MULTILINE`—Makes the ^ and $ to match the start and end of each line. That is, it matches even after and before line breaks in the string. By default, ^ and $ matches the start and end of the whole string.

`re.L or re.LOCALE`—Makes the flag `\w` to match all characters that are considered letters in the given current locale settings.

`re.U or re.UNICODE`—Treats all letters from all scripts as word characters.