FILE I/O PROCESSING AND REGULAR EXPRESSIONS

CHAPTER 4

OPENING AND CLOSING FILES

- Python's built-in open() function is used to open a file stored on a computer hard disk or in the cloud. Here's its syntax:

```
file object = open(file_name [, access_mode][, buffering])
```

No.	Modes	Description
1	r	Opens a file for reading only; the default mode
2	rb	Opens a file for reading only in binary format
3	r+	Opens a file for both reading and writing
4	rb+	Opens a file for both reading and writing in binary format
5	w	Opens a file for writing only
6	wb	Opens a file for writing only in binary format
7	w+	Opens a file for both writing and reading
8	wb+	Opens a file for both writing and reading in binary format
9	a	Opens a file for appending
10	ab	Opens a file for appending in binary format
11	a+	Opens a file for both appending and reading
12	ab+	Opens a file for both appending and reading in binary format

READING AND WRITING TO FILES

- The file.write() method is used to write to a fil, and the file.read() method is used to read data from an opened file.
- A file can be opened for writing (W), reading (r), or both (r+).
- The rename() method is used to rename a file; it takes two arguments: the current filename and the new filename.
- Also, the remove() method can be used to delete files by supplying the name of the file to be deleted as an argument.

DIRECTORIES IN PYTHON

```
In [35]: import os
         os.mkdir("Data 1") # create a directory
         os.mkdir("Data_2")
         os.chdir("Data_3")     # create a Childe directory
         os.getcwd()            # Get the current working
                                  directory

         os.rmdir('Data 1') # remove a directory
         os.rmdir('Data_3') # remove a directory
```

REGULAR EXPRESSIONS

- A regular expression is a special sequence of characters that helps find other strings or sets of strings matching specific patterns; it is a powerful language for matching text patterns.

REGULAR EXPRESSION PATTERNS

No.	Pattern	Description
1	^	Matches beginning of the line.
2	$	Matches end of the line.
3	.	Matches any single character except a newline.
4	[...]	Matches any single character in brackets.
5	[^...]	Matches any single character not in brackets.
6	re*	Matches zero or more occurrences of the preceding expression.

7	`re+`	Matches one or more occurrence of the preceding expression.
8	`re?`	Matches zero or one occurrence of the preceding expression.
9	`re{ n}`	Matches exactly *n* number of occurrences of the preceding expression.
10	`re{ n,}`	Matches *n* or more occurrences of the preceding expression.
11	`re{ n, m}`	Matches at least *n* and at most *m* occurrences of the preceding expression.
12	`a| b`	Matches either *a* or *b*.
13	`(re)`	Groups regular expressions and remembers matched text.
14	`(?imx)`	Temporarily toggles on *i*, *m*, or *x* options within a regular expression.
15	`(?-imx)`	Temporarily toggles off *i*, *m*, or *x* options within a regular expression.
16	`(?: re)`	Groups regular expressions without remembering matched text.
17	`(?imx: re)`	Temporarily toggles on *i*, *m*, or *x* options within parentheses.

(continued)

No.	Pattern	Description
18	`(?-imx: re)`	Temporarily toggles off *i*, *m*, or *x* options within parentheses.
19	`(?#...)`	Comment.
20	`(?= re)`	Specifies the position using a pattern. Doesn't have a range.
21	`(?! re)`	Specifies the position using pattern negation. Doesn't have a range.
22	`(?> re)`	Matches independent pattern without backtracking.
23	`\w`	Matches word characters.
24	`\W`	Matches nonword characters.
25	`\s`	Matches whitespace. Equivalent to `[\t\n\r\f]`.
26	`\S`	Matches nonwhitespace.
27	`\d`	Matches digits. Equivalent to `[0-9]`.
28	`\D`	Matches nondigits.

29	\A	Matches beginning of the string.
30	\Z	Matches end of the string. If a newline exists, it matches just before the newline.
31	\z	Matches end of the string.
32	\G	Matches point where the last match finished.
33	\b	Matches word boundaries when outside brackets.
34	\B	Matches nonword boundaries.
35	\n, \t, etc.	Matches newlines, carriage returns, tabs, etc.
36	\1...\9	Matches nth grouped subexpression.
37	\10	Matches nth grouped subexpression if it matched already.

SPECIAL CHARACTER CLASSES

No.	Example	Description
1	.	Matches any character except newline
2	\d	Matches a digit: [0-9]
3	\D	Matches a nondigit: [^0-9]
4	\s	Matches a whitespace character: [\t\r\n\f]
5	\S	Matches nonwhitespace: [^ \t\r\n\f]
6	\w	Matches a single word character: [A-Za-z0-9_]
7	\W	Matches a nonword character: [^A-Za-z0-9_]

REPETITION CLASSES

No.	Example	Description
1	`ruby?`	Matches `"rub"` or `"ruby"`; the *y* is optional
2	`ruby*`	Matches `"rub"` plus zeros or more *y*s
3	`ruby+`	Matches `"rub"` plus one or more *y*s
4	`\d{3}`	Matches exactly three digits
5	`\d{3,}`	Matches three or more digits
6	`\d{3,5}`	Matches three, four, or five digits

ALTERNATIVES

No	Example	Description
1	python|RLang	Matches "python" or " RLang "
2	R(L|Lang))	Matches " RL" or " RLang"
3	Python(!+|\?)	"Python" followed by one or more ! or one ?

ANCHORS

No.	Example	Description
1	`^Python`	Matches `"Python"` at the start of a string or internal line
2	`Python$`	Matches `"Python"` at the end of a string or line
3	`\APython`	Matches `"Python"` at the start of a string
4	`Python\Z`	Matches `"Python"` at the end of a string
5	`\bPython\b`	Matches `"Python"` at a word boundary
6	`\brub\B`	`\B` is nonword boundary: matches `"rub"` in *rube* and *ruby* but not on its own
7	`Python(?=!)`	Matches `"Python,"` if followed by an exclamation point
8	`Python(?!!)`	Matches `"Python,"` if not followed by an exclamation point

SUMMARY

- The chapter covered how to open files for reading, writing, or both. Furthermore, it covered how to access the attributes of open files and close all opened sessions.
- The chapter covered how to collect data directly for users via the screen.
- It covered regular expressions and their patterns and special character usage.
- The chapter covered how to apply regular expressions to extract data and how to use alternatives, anchors, and repetition expressions for data extraction.

EXERCISE