Table of Contents

Chapter 16

RegEx

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern.
Regex can be used to add, remove, isolate, and manipulate all kinds of text and data. It could be used as a simple text editor command, e.g., search and replace, or as it's own powerful text-processing language.

RegEx Module:

Python has a built-in package called re, which can be used to work with Regular Expressions.

```
# import the re module:
import re
```

RegEx in Python:

When you have imported the re module, you can start using regular expressions.
Ex.

```
import re
#Check if the string starts with "Madi" and ends with "perfect":
txt = "Madi is perfect"
x = re.search("^Madi.*perfect$", txt)
if x:
  print("YES! We have a match!")
else:
  print("No match")
# YES! We have a 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
compile	combine a regular expression pattern into pattern objects, which can be used for pattern matching.

Metacharacters:

Metacharacters are part of regular expression and are the special characters that symbolize regex patterns or formats. Every character is either a metacharacter or a regular character in a regular expression. However, metacharacters have a special meaning.

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

Sets:

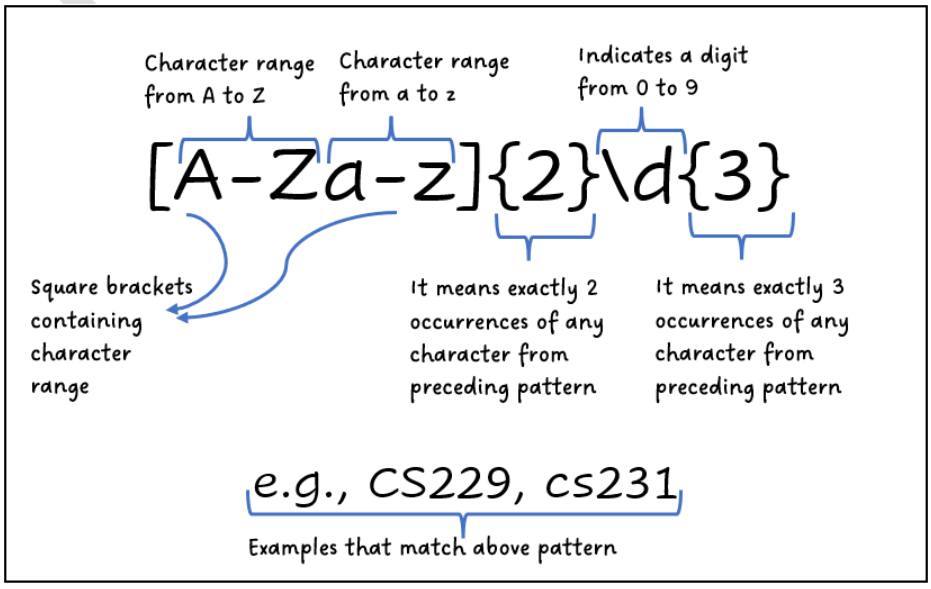
Set is a set of characters inside a pair of square brackets [] with a special meaning. A character set (or a character class) is a set of characters, for example, digits (from 0 to 9), alphabets (from a to z), and whitespace. A character set allows you to construct regular expressions with patterns that match a string with one or more characters in a set.

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

Special Sequence:

The special sequence represents the basic predefined character classes, which have a unique meaning. Each special sequence makes specific common patterns more comfortable to use. For example, you can use \d sequence as a simplified definition for character class [0-9], which means match any digit from 0 to 9.
A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

Character	Description
\A	Returns a match if the specified characters are at the beginning of the string
\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")
\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")
\d	Returns a match where the string contains digits (numbers from 0-9)
\D	Returns a match where the string DOES NOT contain digits
\s	Returns a match where the string contains a white space character
\S	Returns a match where the string DOES NOT contain a white space character
\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	Returns a match where the string DOES NOT contain any word characters
\Z	Returns a match if the specified characters are at the end of the string



Be Practical:
import the re module:
import re

- The findall() Function:
The findall() function returns a list containing all matches.
Ex.
```
import re
#Return a list containing every occurrence of "an":
txt = '''Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.'''
x = re.findall("an", txt)
print(x)  # ['an', 'an', 'an', 'an', 'an', 'an']
```

- The search() Function:
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.
Ex.
```
import re
txt = '''Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.'''
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
# The first white-space character is located in position: 6
# If no matches are found, the value None is returned.
```

- The split() Function:
The split() function returns a list where the string has been split at each match.
Ex.
```
import re
#Split the string at every white-space character:
txt = '''Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.'''
x = re.split("\s", txt)
```

```
print(x)
# ['Python', 'is', 'an', 'experiment', 'in', 'how', 'much', 'freedom', 'programmers', 'need.', 'Too', 'much', 'freedom', 'and', 'nobody', 'can', 'read', "another's", 'code;', 'too', 'little', 'and', 'expressiveness', 'is', 'endangered.']
#You can control the number of occurrences by specifying the maxsplit parameter.
# x = re.split("\s", txt, 1)
# ['Python', 'is', 'an', 'experiment', 'in', 'how', 'much', 'freedom', 'programmers', 'need.', "Too much freedom and nobody can read another's code; too little and expressiveness is endangered."]
```

-The sub() Function:

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

Ex.

```
import re
#Replace all white-space characters with backslash "/":
txt = '''Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.'''
x = re.sub("\s",r"/", txt)
print(x)
#
Python/is/an/experiment/in/how/much/freedom/programmers/need./Too/much/freedom/and/nobody/can/read/another's/code;/too/little/and/expressiveness/is/endangered.

# You can control the number of replacements by specifying the count parameter:
# x = re.sub("\s",r"/", txt,7)
# print(x)
# Python/is/an/experiment/in/how/much/freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.
```

- compile Function:

Python’s re.compile() method is used to compile a regular expression pattern provided as a string into a regex pattern object (re.Pattern). Later we can use this pattern object to search for a match inside different target strings using regex methods such as a re.match() or re.search().

Syntax:

```
re.compile(pattern, flags=0)
```

pattern: regex pattern in string format, which you are trying to match inside the target string.
flags: The expression’s behavior can be modified by specifying regex flag values. This is an optional parameter.
Ex.

```
import re

# Target String one
str1 = "Madi's luck numbers are 555 75 777"

# pattern to find three consecutive digits
string_pattern = r"\d{3}"
# compile string pattern to re.Pattern object
regex_pattern = re.compile(string_pattern)

# print the type of compiled pattern
print(type(regex_pattern))
# Output <class 're.Pattern'>

# find all the matches in string one
result = regex_pattern.findall(str1)
print(result)
# Output ['555', '777']

# Target String two
str2 = "Neuro's luck numbers are 111 212 415"
# find all the matches in second string by reusing the same pattern
result = regex_pattern.findall(str2)
print(result)
# Output ['111', '212', '415']
```

Metacharacters Examples:

Ex1.

```
import re
txt = "Madi Is Perfect"
#Find all lower case characters alphabetically between "a" and "m":
x = re.findall("[a-m]", txt)
print(x) # ['a', 'd', 'i', 'e', 'f', 'e', 'c']
```

Ex2.

```
import re
txt = "Madi is perfect"
#Check if the string starts with 'Madi':
x = re.findall("^Madi", txt)
if x:
  print("Yes, the string starts with 'Madi'") # Yes, the string starts with 'Madi'
else:
  print("No match")
```

Ex3.

```
import re
txt = "Madi is Perfect"
#Search for a sequence that starts with "Pe", followed by 0 or more  (any) characters,
and an "t":
x = re.findall("Pe.*t", txt)
print(x) # ['Perfect']
```

Ex4.

```
import re
txt = "Madi is Perfect"
#Search for a sequence that starts with "he", followed excactly 2 (any) characters, and
an "t":
x = re.findall("Perf.{2}t", txt)
print(x) # ['Perfect']
```

Ex5.

```
import re
txt = "Madi is Perfect"
#Check if the string contains either "Perfect" or "Feeling":
x = re.findall("Perfect|Feeling", txt)
print(x) # ['Perfect']
```

Special Sequences:

Ex1.

```
import re
txt = "Madi is perfect"
#Check if the string starts with "Madi":
x = re.findall("\AMadi", txt)
print(x)
if x:
  print("Yes, there is a match!")
else:
  print("No match")
# ['Madi']
# Yes, there is a match!
```

Ex2.

```
import re
txt = "Madi is perfect"
#Return a match at every white-space character:
x = re.findall("\s", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
# [' ', ' ']
# Yes, there is at least one match!
```

Ex3.

```
import re
txt = "Madi is perfect"
#Return a match at every NON word character (characters NOT between a and Z. Like
"!", "?" white-space etc.):
x = re.findall("\W", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
# [' ', ' ']
# Yes, there is at least one match!
```

Sets:

Ex1.

```
import re
txt = "I can do this all day"
#Check if the string has any a, r, or n characters:
x = re.findall("[arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
# ['a', 'n', 'a', 'a']
# Yes, there is at least one match!
```

Ex2.

```
import re
txt = "Whatever it takes"
#Check if the string has other characters than a, r, or n:
x = re.findall("[^arn]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
# ['W', 'h', 't', 'e', 'v', 'e', ' ', 'i', 't', ' ', 't', 'k', 'e', 's']
# Yes, there is at least one match!
```

Ex3.

```
import re
txt = "8 times before 11:45 AM"
#Check if the string has any two-digit numbers, from 00 to 59:
x = re.findall("[0-5][0-9]", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
# ['11', '45']
# Yes, there is at least one match!
```

Match Object:

A Match Object is an object containing information about the search and the result.
If there is no match, the value None will be returned, instead of the Match Object.

Ex.

```
import re
#The search() function returns a Match object:
txt = "I was wondering why the frisbee kept getting bigger and bigger, but then it hit me."
x = re.search("on", txt)
print(x)
# <re.Match object; span=(7, 9), match='on'>
print(txt[7:9]) # on
```

Match Object properties and Methods:

The Match object has properties and methods used to retrieve information about the search, and the result:
.span() returns a tuple containing the start-, and end positions of the match.
.string returns the string passed into the function
.group() returns the part of the string where there was a match

.span():

Ex.

```
import re
#Search for an upper case "I" character in the beginning of a word, and print its position:
txt = "It is what it is"
x = re.search(r"\bI\w+", txt)
print(x.span())
# (0, 2)
```

.string:

Ex.

```
import re
#The string property returns the search string:
txt = "It's Me Deadpool"
x = re.search(r"\bD\w+", txt)
print(x.string)
# It's Me Deadpool
```

.group():
Ex.

```
import re
#Search for an upper case "S" character in the beginning of a word, and print the word:
txt = "The Hardest Choices Require The Strongest Wills."
x = re.search(r"\bS\w+", txt)
print(x.group())
# Strongest
```

More RegEx. Examples:
Ex.

```
import re
phone_number = "+91-1234567890"
# Matches Indian phone numbers with or without the country code (+91) and with 10 digits
pattern = r'^(\+91[\-\s]?)?[0]?(91)?[6789]\d{9}$'
if re.match(pattern, phone_number):
   print("Valid phone number")
else:
   print("Invalid phone number")
# Valid Phone Number
```

Explanation:
- ^ matches the start of the string
- (\+91[\-\s]?)? matches the optional country code (+91) and an optional hyphen or space character
- [0]? matches an optional 0 digit
- (91)? matches an optional 91 digit (used for landlines)
[6789]\d{9} matches 10 digits starting with 6, 7, 8, or 9
- '$ matches the end of the string

Ex.

```
import re
email = "example@gmail.com"
# Matches Gmail email addresses
pattern = r'^[a-zA-Z0-9._%+-]+@gmail\.com$'
if re.match(pattern, email):
   print("Valid Gmail email address")
else:
   print("Invalid Gmail email address")
# Valid Gmail email address
```

Explanation:
- ^ matches the start of the string
- [a-zA-Z0-9._%+-]+ matches one or more characters that are letters, digits, or any of the special characters ., _, %, +, or -
- @gmail\.com matches the literal characters "@gmail.com"
- $ matches the end of the string

Ex.

```
import re
ip_address = "192.168.0.1"
# Matches IP addresses in the format xxx.xxx.xxx.xxx where xxx is a number between 0 and 255

pattern = r'^([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'

if re.match(pattern, ip_address):
   print("Valid IP address")
else:
   print("Invalid IP address")

#Valid IP address
```

Explanation:
- ^ matches the start of the string
- ([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]) matches a number between 0 and 255
- \. matches a literal period character (escaped with a backslash)
- The previous two steps are repeated three times to match three more octets (groups of three digits)
- $ matches the end of the string