Python RegEx

Try these examples - Online Python Compiler (Interpreter) (programiz.com)

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 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:

Example

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
```

```
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
```

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

Metacharacters are characters with a special meaning:

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

The findall() Function

The `findall()` function returns a list containing all matches.

Example

Print a list of all matches:

```
import re

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

If no matches are found, an empty list is returned:

Example

Return an empty list if no match was found:

```
import re

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

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:

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())
```

If no matches are found, the value `None` is returned:

Example

Make a search that returns no match:

```
import re

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

The split() Function

The `split()` function returns a list where the string has been split at each match:

Example

Split at each white-space character:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

You can control the number of occurrences by specifying the `maxsplit` parameter:

Example

Split the string only at the first occurrence:

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt, 1)
print(x)
```

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 = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)
```

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

Example

Replace the first 2 occurrences:

```
import re

txt = "The rain in Spain"
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 = "The rain in Spain"
x = re.search("ai", txt)
print(x) #this will print an object
```

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

Example

Print the position (start- and end-position) of the first match occurrence.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

Example

Print the string passed into the function:

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

Example

Print the part of the string where there was a match.

The regular expression looks for any words that starts with an upper case "S":

```
import re

txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.group())
```

- Now, if they ask you to code using regular expression- you can code a simple password validation program in python using regular expression.

Example 1-

Python Program using regular expression- Using regex compile() method of Regex module makes a Regex object, making it possible to execute regex functions onto the *pat* variable. Then we check if the pattern defined by *pat* is followed by the input string *passwd*. If so, the search method returns *true*, which would allow the password to be valid.

(Count the characters using regex)

```
import re


string = "ThisIsGeeksforGeeks !, 123"

# Creating separate lists using
# the re.findall() method.
uppercase_characters = re.findall(r"[A-Z]", string)
lowercase_characters = re.findall(r"[a-z]", string)
numerical_characters = re.findall(r"[0-9]", string)
special_characters = re.findall(r"[, .!?]", string)

print("The no. of uppercase characters is", len(uppercase_characters))
print("The no. of lowercase characters is", len(lowercase_characters))
print("The no. of numerical characters is", len(numerical_characters))
print("The no. of special characters is", len(special_characters))
```

-Password Validation: Using regex compile() method of Regex module makes a Regex object, making it possible to execute regex functions onto the *pat* variable. Then we check if the pattern defined by *pat* is followed by the input string *passwd*. If so, the search method returns *true*, which would allow the password to be valid.

```
import re

def main():
    passwd = 'Geek12@'
    reg = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-
z\d@$!#%*?&]{6,20}$"

    # compiling regex
    pat = re.compile(reg)
```

```
    # searching regex
    mat = re.search(pat, passwd)

    # validating conditions
    if mat:
        print("Password is valid.")
    else:
        print("Password invalid !!")

# Driver Code
if __name__ == '__main__':
    main()
```

1. `Explanation -` The `re` module is imported to work with regular expressions.
2. The `main()` function is defined. This function serves as the entry point for the script.
3. The variable `passwd` contains the password to be validated. In this case, it is set to `'Geek12@'`.
4. The regular expression pattern `reg` is defined. It uses lookaheads to enforce the following conditions for the password:
    - At least one lowercase letter: `(?=.*[a-z])`
    - At least one uppercase letter: `(?=.*[A-Z])`
    - At least one digit: `(?=.*¥d)`
    - At least one special character from `@$!%*#?&`: `(?=.*[@$!%*#?&])`
    - Allowed characters for the password (6 to 20 characters): `[A-Za-z¥d@$!#%*?&]{6,20}`
5. The regular expression pattern `pat` is compiled using `re.compile(reg)`.
6. The `re.search()` function is used to search for a match between the pattern and the `passwd` string. The result is stored in the `mat` variable.
7. The code checks whether a match was found (`if mat`). If a match is found, it means the password is valid, and "Password is valid." is printed. Otherwise, "Password invalid !!" is printed.
8. Finally, the script checks if it is being run as the main module (`if __name__ == '__main__':`) and calls the `main()` function.

Example 2-

```
# importing the module
import re
```

```
# opening and reading the file
with open('C:/Users/user/Desktop/New Text Document.txt') as fh:
   fstring = fh.readlines()

# declaring the regex pattern for IP addresses
pattern = re.compile(r'(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})')

# initializing the list object
lst=[]

# extracting the IP addresses
for line in fstring:
   lst.append(pattern.search(line)[0])

# displaying the extracted IP addresses
print(lst)
```

1. Explanation- The `re` module is imported to work with regular expressions.
2. The code opens a text file using the `open()` function. The file path `'C:/Users/user/Desktop/New Text Document.txt'` should be modified to the actual path and name of your text file.
3. The `readlines()` method is used to read the contents of the file and store each line as a separate string in the `fstring` variable.
4. A regular expression pattern for matching IP addresses is defined using the `re.compile()` function. The pattern `r'(¥d{1,3}¥.¥d{1,3}¥.¥d{1,3}¥.¥d{1,3})'` matches a series of four numbers separated by periods, where each number can be 1 to 3 digits long.
5. An empty list `lst` is initialized to store the extracted IP addresses.
6. The code iterates over each line in `fstring` using a for loop. For each line, it searches for a match to the IP address pattern using `pattern.search(line)`.
7. The match object returned by `pattern.search(line)` is accessed using `[0]` to retrieve the actual matched IP address.
8. The extracted IP address is appended to the `lst` list.
9. After iterating through all the lines, the `lst` list contains all the extracted IP addresses.
10. Finally, the code prints the `lst` list, which displays the extracted IP addresses.

To use this code, make sure you have a text file with IP addresses in the specified format. Update the file path in `open()` to the location of your text file. When you run the script, it will extract the IP addresses from the file and print them.