| **Meta character** | **Description** |
| --- | --- |
| \ | Marks the next character as either a special character or a literal. For example, n matches the character *n*, whereas \n matches a newline character. The sequence \\ matches \ and \( matches (. |
| ^ | Matches the beginning of input. |
| $ | Matches the end of input. |
| \* | Matches the preceding character zero or more times. For example, zo\* matches either *z* or *zoo*. |
| + | Matches the preceding character one or more times. For example, zo+ matches *zoo* but not *z*. |
| ? | Matches the preceding character zero or one time. For example, a?ve? matches the *ve* in *never*. |
| . | Matches any single character except a newline character. |
| (pattern) | Matches a pattern and remembers the match. The matched substring can be retrieved from the resulting matches collection by using this code: Item [0]...[n]. To match parentheses characters ( ), use \( or \). |
| *x*|*y* | Matches either *x* or*y*. For example, z|wood matches *z* or *wood*. (z|w)oo matches *zoo* or *wood*. |
| {*n*} | *n* is a non-negative integer. Matches exactly *n* times. For example, o{2} does not match the *o* in *Bob*, but matches the first two *o*s in *foooood*. |
| {*n*,} | In this expression, *n* is a non-negative integer. Matches the preceding character at least *n* times. For example, o{2,} does not match the *o* in *Bob* and matches all the *o*s in *foooood*. The o{1,} expression is equivalent to o+ and o{0,} is equivalent to o\*. |
| {*n*,*m*} | The *m* and *n* variables are non-negative integers. Matches the preceding character at least *n* and at most *m* times. For example, o{1,3} matches the first three *o*s in *fooooood*. The o{0,1} expression is equivalent to o?. |
| [*xyz*] | A character set. Matches any one of the enclosed characters. For example, [abc] matches the *a* in *plain*. |
| [^*xyz*] | A negative character set. Matches any character that is not enclosed. For example, [^abc] matches the *p* in *plain*. |
| [*a-z*] | A range of characters. Matches any character in the specified range. For example, [a-z] matches any lowercase alphabetic character in the English alphabet. |
| [^*m-z*] | A negative range of characters. Matches any character that is not in the specified range. For example, [m-z] matches any character that is not in the range *m* through *z*. |
| \A | Matches only at beginning of a string. |
| \b | Matches a word boundary, that is, the position between a word and a space. For example, er\b matches the *er* in *never* but not the *er* in *verb*. |
| \B | Matches a nonword boundary. The ea\*r\B expression matches the *ear* in *never early*. |
| \d | Matches a digit character. |
| \D | Matches a non-digit character. |
| \f | Matches a form-feed character. |
| \n | Matches a newline character. |
| \r | Matches a carriage return character. |
| \s | Matches any white space including spaces, tabs, form-feed characters, and so on. |
| \S | Matches any non-white space character. |
| \t | Matches a tab character. |
| \v | Matches a vertical tab character. |
| \w | Matches any word character including underscore. This expression is equivalent to [A-Za-z0-9\_]. |
| \W | Matches any non-word character. This expression is equivalent to [^A-Za-z0-9\_]. |
| \z | Matches only the end of a string. |
| \Z | Matches only the end of a string, or before a newline character at the end. |

**Code With Harry Overview**

### Regular Expressions | Python Tutorials For Absolute Beginners In Hindi #86

***Regular expressions*** are used to perform search-related tasks in Python. In this tutorial, our primary focus should be on understanding because we are going to cover a concept that has a wide range of uses. To work with regular expressions, we have to import a built-in module in Python called **‘re’.**

import re

Copy

The module defines several functions and constants to work with RegEx. The “re” module is composed of five functions known as:

* **findall**: It finds all searches for matches and prints resultant in the form of a list.
* **search**: It works the same as a findall, but the resultant is a matched object if any is found.
* **split**: The split function splits the string from every matched into two new strings.
* **sub**: The sub-function works exactly like a replace function in notepad or MS Word. It replaces the original word with a word of our choice.
* **finditer**: The finditer yields an iterator as a resultant with all the objects that match the one we sent it) finditer supports more attributes than any other function defined above. It also provides more details related to the matched object. So, most of the examples we are going to see next will contain a finditer function in them.

So, you must be wondering that all of the searchings can easily be done using a simple loop with some conditions so, what is the purpose of the “re” module. Well “re” module is used for complex searching, using Metacharacters and special sequences.

Metacharacters have special meaning in Python, and they are used with “re” modules to search for keywords and objects more technically and efficiently. We will see the working of a few Meta Characters in this tutorial so you can get an idea. I will provide you with a list of these characters and their working at the end of this tutorial.

#### Use of “^”:-

We use the” ^” symbol to check whether the string is starting from the keyword we wrote after ^ or not. For example, if a string starts from CodeWithHarry and we are searching the keyword using ^CodeWithHarry with finditer, it will return us whether our string is starting from the searched keyword or not. The same is the case for $ sign. It will check whether our string is ending with the specific keyword or not.

#### Use of “|”:-

We can also use a unique character “|” to use more than one condition, so if we use it for the above case, then it will check whether the string starts or ends with CodeWithHarry.Now we will move on to special sequences. We will see a few special sequences in this tutorial, and you can have a look at the list of these sequences at the end of the tutorial description for further practice.

* **\A:**         the resultant is a match if the input characters are at the beginning of the string
* **\b**          the resultant is a match whether the input characters are at the beginning or the end of a word
* **\d**          the resultant is a match if the string contains any digits
* **\s**           the resultant is a match if the string contains a white space character

There are many metacharacters supported by the re module. Some characters with their working are the following:

* ‘**.’:**Matches any single character except newline
* **‘$’:**Anchors a match at the end of a string
* ‘**\*’:**Matches zero or more repetitions
* ‘**+':**Matches one or more repetitions
* ‘**{}’:**Matches an explicitly specified number of repetitions
* ‘**[]’:**Specifies a character class

To explore more about the re module, check the <https://docs.python.org/3/library/re.html> python documentation.

#### 

#### Re.txt file as described in the video!

Meta Characters  
[] 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  
{} Exactly the specified number of occurrences  
| Either or  
() Capture and group  
Special Sequences  
\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 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  
  
\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