BCA – 502: Python Programming

Rahul Kumar Singh

In today's Class we have discussed on Powerful pattern matching and searching in Python and Writing log file in python.

Pattern matching in Python with Regex:-

A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are widely used in UNIX world.

The Python module re provides full support for Perl-like regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

We would cover two important functions, which would be used to handle regular expressions. But a small thing first: There are various characters, which would have special meaning when they are used in regular expression. To avoid any confusion while dealing with regular expressions, we would use Raw Strings as r'expression'.

The match Function:-

This function attempts to match RE pattern to string with optional flags.

Syntax:- Here is the syntax for this function –

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

The re.search() method takes two arguments, a regular expression pattern and a string and searches for that pattern within the string. If the pattern is found within the string, search() returns a match object or None otherwise. So in a regular expression, given a string, determine whether that string matches a given pattern, and, optionally, collect substrings that contain relevant information. A regular expression can be used to answer questions like –

- Is this string a valid URL?
- Which users in /etc/passwd are in a given group?
- What is the date and time of all warning messages in a log file?
- What username and document were requested by the URL a visitor typed?

Matching patterns:-

Regular expressions are complicated mini-language. They rely on special characters to match unknown strings, but let's start with literal characters, such as letters, numbers, and the space character, which always match themselves.

Example:-1

```
#!/usr/bin/python

import re
```

```
line = "Cats are smarter than dogs"
matchObj = re.match( r'(.*) are (.*?) .*', line, re.M|re.I)
if matchObj:
   print "matchObj.group() : ", matchObj.group()
   print "matchObj.group(1) : ", matchObj.group(1)
   print "matchObj.group(2) : ", matchObj.group(2)
else:
   print "No match!!"
```

Output:-

```
matchObj.group() :  Cats are smarter than dogs
matchObj.group(1) :  Cats
matchObj.group(2) :  smarter
```

Example:-2

```
#Need module 're' for regular expression
import re
search_string = "RegularExpression"
pattern = "Regular"
match = re.match(pattern, search_string)
#If-statement after search() tests if it succeeded
```

```
if match:
  print("regex matches: ", match.group())
else:
  print('pattern not found')
```

Output:-

```
regex matches:  Regular
```

Example:-3

```
#!/usr/bin/python
import re
line = "Cats are smarter than dogs";
matchObj = re.match( r'dogs', line, re.M|re.I)
if matchObj:
   print "match --> matchObj.group() : ", matchObj.group()
else:
   print "No match!!"
searchObj = re.search( r'dogs', line, re.M|re.I)
if searchObj:
   print "search --> searchObj.group() : ", searchObj.group()
```

```
else:
   print "Nothing found!!"
```

Output:-

When the above code is executed, it produces the following result –

```
No match!!
search --> searchObj.group() :  dogs
```

Matching a string:-

The “re” module of python has numerous method, and to test whether a particular regular expression matches a specific string, you can use re.search(). The re.MatchObject provides additional information like which part of the string the match was found.

Syntax:-

```
matchObject = re.search(pattern, input_string, flags=0)
```

Example:-

```
import re
# Lets use a regular expression to match a date string.
regex = r"([a-zA-Z]+) (\d+)"
if re.search(regex, "Jan 2"):
```

```
    match = re.search(regex, "Jan 2")
    # This will print [0, 5), since it matches at the beginning and end of the
    # string
    print("Match at index %s, %s" % (match.start(), match.end()))
    # The groups contain the matched values.  In particular:
    #  match.group(0) always returns the fully matched string
    #match.group(1), match.group(2)...will return the capture
    #   groups in order from left to right in the input string
    #    match.group() is equivalent to match.group(0)
    # So this will print "Jan 2"
    print("Full match: %s" % (match.group(0)))
    # So this will print "Jan"
    print("Month: %s" % (match.group(1)))
    # So this will print "2"
    print("Day: %s" % (match.group(2)))
else:
    # If re.search() does not match, then None is returned
    print("Pattern not Found! ")
```

Writing log files in python:-

Logging is a means of tracking events that happen when some software runs. The software's developer adds logging calls to their code to indicate that certain events have occurred. An event is described by a descriptive message which can optionally contain variable data (i.e. data that is potentially different for each occurrence of the event). Events also have an importance which the developer ascribes to the event; the importance can also be called the level or severity.

When to use logging:-

Logging provides a set of convenience functions for simple logging usage. These are debug(), info(), warning(), error() and critical(). To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it.

Create a logging file:-

To create a logging file First of all, we need to import the logging module, followed by using the logger to check the current status and log messages. We are having 5 severity levels namely –

- Warning
- Info
- Error

- Critical
- Debug

DEBUG:- Used to give detailed information. This level is mostly used for diagnosing issues in code.

INFO:- Confirms the program works as expected.

WARNING:- An indication that an unexpected event occured or may occur.

ERROR:- Serious issue. Indicates that a program was unable to perform some action due to an error.

CRITICAL:- A serious error. Indicates that the program may be unable to continue running.

There is a built-in module called **logging** for creating a log file in Python.

To start logging messages to a file in Python, you need to configure the logging module.

To configure the logging module, call **logging.basicConfig()**

Example:-

```
import logging

logging.basicConfig(filename="log.txt",
level=logging.DEBUG)
```

```
logging.debug("Debug logging test...")

logging.info("Program is working as expected")

logging.warning("Warning, the program may not function properly")

logging.error("The program encountered an error")

logging.critical("The program crashed")
```

Output:-

```
DEBUG:root:Debug logging test...

DEBUG:root:Debug logging test...

INFO:root:Program is working as expected

WARNING:root:Warning, the program may not function properly

ERROR:root:The program encountered an error

CRITICAL:root:The program crashed
```