Python: Date and Time

Python Date and Time

There is a popular time module available in Python which provides functions for working with times, and for converting between representations. The function time.time() returns the current system time in ticks since 00:00:00 hrs January 1, 1970(epoch).

```
import time;  # This is required to include time module.

ticks = time.time()
print(ticks)
```

Getting current time

```
import time;

localtime = time.localtime(time.time())
print(localtime)
```

Getting formatted time

```
import time;

localtime = time.asctime( time.localtime(time.time()) )
print(localtime)
```

Getting calendar for a month

```
import calendar

cal = calendar.month(2020, 12)
print(cal)
```

The calendar module supplies calendar-related functions, including functions to print a text calendar for a given month or year.

By default, calendar takes Monday as the first day of the week and Sunday as the last one. To change this, call **calendar.setfirstweekday()** function.

calendar.calendar(year,w=2,l=1,c=6)
Returns a multiline string with a calendar for year year formatted into three columns separated by c spaces. w is the width in characters of each date; each line has length 21*w+18+2*c. l is the number of lines for each week.

calendar.firstweekday( )
Returns the current setting for the weekday that starts each week. By default, when calendar is first imported, this is 0, meaning Monday.

calendar.isleap(year)
Returns True if year is a leap year; otherwise, False.

calendar.leapdays(y1,y2)
Returns the total number of leap days in the years within range(y1,y2).

calendar.month(year,month,w=2,l=1)
Returns a multiline string with a calendar for month month of year year, one line per week plus two header lines. w is the width in characters of each date; each line has length 7*w+6. l

Python: RegEx

Python RegEx

A **Reg**ular **Ex**pression (RegEx) is a sequence of characters that defines a search pattern.

Python has a module named **re** to work with RegEx.

```
import re
pattern = '^a...s$'
test_string = ‘alias'
result = re.match(pattern, test_string)

if result:
  print("Search successful.")
else:
  print("Search unsuccessful.")
```

Meta Characters

Metacharacters are characters that are interpreted in a special way by a RegEx.

[] . ^ $ * + ? {} () \ |
Square brackets specifies a set of characters you wish to match.
You can also specify a range of characters using - inside square brackets.

[a-e] is the same as [abcde].
[1-4] is the same as [1234].

You can complement (invert) the character set by using caret ^ symbol at the start of a square-bracket.

[^abc] means any character except a or b or c.
[^0-9] means any non-digit character.

. - Period
A period matches any single character (except newline '\n’).

^ - Caret
The caret symbol ^ is used to check if a string starts with a certain character.

$ - Dollar
The dollar symbol $ is used to check if a string ends with a certain character.

* - Star
The star symbol * matches zero or more occurrences of the pattern left to it.

+ - Plus
The plus symbol + matches one or more occurrences of the pattern left to it.

? - Question Mark
The question mark symbol ? matches zero or one occurrence of the pattern left to it.

{} - Braces
Consider this code: {n,m}. This means at least n, and at most m repetitions of the pattern left to it.

| - Alternation
Vertical bar | is used for alternation (or operator).

() - Group
Parentheses () is used to group sub-patterns. For example, (a|b|c)xz match any string that matches either a or b or c followed by xz

\ - Backslash
Backlash \ is used to escape various characters including all metacharacters.

\A - Matches if the specified characters are at the start of a string.
\b - Matches if the specified characters are at the beginning or end of a word.
\B - Opposite of \b. Matches if the specified characters are not at the beginning or end of a word.
\d - Matches any decimal digit. Equivalent to [0-9]
\D - Matches any non-decimal digit. Equivalent to [^0-9]
\s - Matches where a string contains any whitespace character.
Equivalent to [\t\n\r\f\v].
\S - Matches where a string contains any non-whitespace character.
Equivalent to [^ \t\n\r\f\v].
\w - Matches any alphanumeric character (digits and alphabets).
Equivalent to [a-zA-Z0-9_]. By the way, underscore _ is also considered an alphanumeric character.
\W - Matches any non-alphanumeric character. Equivalent to [^a-zA-Z0-9_]
\Z - Matches if the specified characters are at the end of a string.

re.findall()

The re.findall() method returns a list of strings containing all matches.

```
import re
string = ‘talent battle 12 20 21 hello'
pattern = '\d+'
result = re.findall(pattern, string)
print(result)
```

If the pattern is not found, re.findall() returns an empty list.

re.split()

The re.split method splits the string where there is a match and returns a list of strings where the splits have occurred.

```
import re
string = ‘Talent Battle:12 Hello:2021.'
pattern = '\d+'

result = re.split(pattern, string)
print(result)
```

```
# Program to remove all whitespaces
import re

# multiline string
string = 'abc 12\
de 23 \n f45 6'

# matches all whitespace characters
pattern = '\s+'

# empty string
replace = ‘’”

new_string = re.sub(pattern, replace, string)
print(new_string)

# Output: abc12de23f456
```

re.search()

The re.search() method takes two arguments: a pattern and a string.

The method looks for the first location where the RegEx pattern produces a match with the string.

If the search is successful, re.search() returns a match object; if not, it returns None.

Syntax:

```
match = re.search(pattern, str)
```

```
import re

string = “Talent Battle is awesome"

match = re.search(‘\ATalent', string)

if match:
  print("pattern found inside the string")
else:
  print("pattern not found")
```

Using r prefix before RegEx

When r or R prefix is used before a regular expression, it means raw string. For example, '\n' is a new line whereas r'\n' means two characters: a backslash \ followed by n.

Backlash \ is used to escape various characters including all metacharacters. However, using r prefix makes \ treat as a normal character.

```
import re

string = '\n and \r are escape sequences.'

result = re.findall(r'[\n\r]', string)
print(result)

# Output: ['\n', '\r']
```