Python Programming Basics

www.huawei.com

HUAWEI

Contents

HUAWEI

Getting the Current Date and Time

- Let’s see how to get the current date and time.

```
>>> from datetime import datetime
>>> now = datetime.now() # Get the current datetime
>>> print(now)
2015-05-18 16:28:07.198690
>>> print(type(now))
<class ‘datetime.datetime’>
```

- Note that datetime is a module, and it also contains a datetime class. Only the class imported by from *datetime import datetime* is the datetime class. If you import only the import datetime, you need to refer to the full name datetime.datetime.

  DateTime.Now () returns the current date and time, with the type datetime.

Getting Formatted Time and Date

Format Time

```
>>>import time
>>>localtime = time.asctime( time.localtime( ))
>>>print("Local time :", localtime)
```

Output:

```
Local time: Thu Apr  7 10:05:21 2016
```

Format Date

```
>>>import time
# Format into 2016-03-20 11:45:39
>>>print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
# Format into Sat Mar 28 22:24:24 2016
>>>print(time.strftime("%a %b %d %H:%M:%S %Y", time.localtime()))
```

Output:
2016-04-07 10:25:09
Thu Apr 07 10:25:09 2016

Converting datetime to timestamp

- In a computer, time is represented by numbers.
- We refer to the January 1, 1970 00:00:00 utc+00:00 time zone as epoch time, which is recorded as 0 (a negative timestamp for before 1970), and the current time is the number of seconds relative to epoch time, called timestamp. You can take that timestamp = 0 = 1970-1-1 00:00:00 utc+0:00. The corresponding Beijing time is: timestamp = 0 = 1970-1-1 08:00:00 utc+8:00. The value of the visible timestamp has nothing to do with the time zone, because once the timestamp is determined, the UTC time is determined, and time in any timezone after conversion is determined. That is why the current time that the computer stores is represented by timestamp. Because timestamps of the computers around the world are the same at any given time, only the timestamp () method needs to be called to convert a datetime type to timestamp.

```
>>> from datetime import datetime
>>> dt = datetime(2015,4,19,12,20)  #Create using specified datetime
>>> dt.timestamp()     #Convert datetime into timestamp
1429417200.0
```

- Note that Python's timestamp is a floating-point number. If there are decimal places, the decimal places represent the number of milliseconds.
- The timestamp of some programming languages, such as Java and JavaScript, uses integers to represent the number of milliseconds, in which case a floating-point representation of Python can only be achieved by dividing the timestamp by 1000.

Converting timestamp to datetime

- To convert timestamp to datetime, use the fromtimestamp () method provided by datetime:

```
>>> from datetime import datetime
>>> t = 1429417200.0
>>> print(datetime.fromtimestamp(t))
2015-04-19 12:20:00
```

- Note that timestamp is a floating-point number, and it does not have the concept of a time zone, but datetime has a time zone. Local time refers to the time zone set by the current operating system. For example, the Beijing time zone is East 8, then local time: 2015-04-19 12:20:00. That is, the time of the utc+8:00 time zone: 2015-04-19 12:20:00 utc+8:00. At the moment Greenwich Standard Time and Beijing time offset by 8 hours. That is, time in utc+0:00 time zone should be: 2015-04-19 04:20:00 utc+0:00. Timestamp can also be converted directly to time in the UTC standard time zone:

```
>>> from datetime import datetime
>>> t = 1429417200.0
>>> print(datetime.fromtimestamp(t)) # local time
2015-04-19 12:20:00
>>> print(datetime.utcfromtimestamp(t)) # UTC time
2015-04-19 04:20:00
```

Get Calendar of a Month

- The calendar module can process yearly calendars and monthly calendars using multiple methods, for example, printing a monthly calendar.

```
>>>import calendar
>>>cal = calendar.month(2014, 4)
>>>print("output calendar of April 2014:")
>>>print(cal)
Output:
The following is the calendar of April 2014
      April 2014
Mo Tu We Th Fr Sa Su
       1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Contents

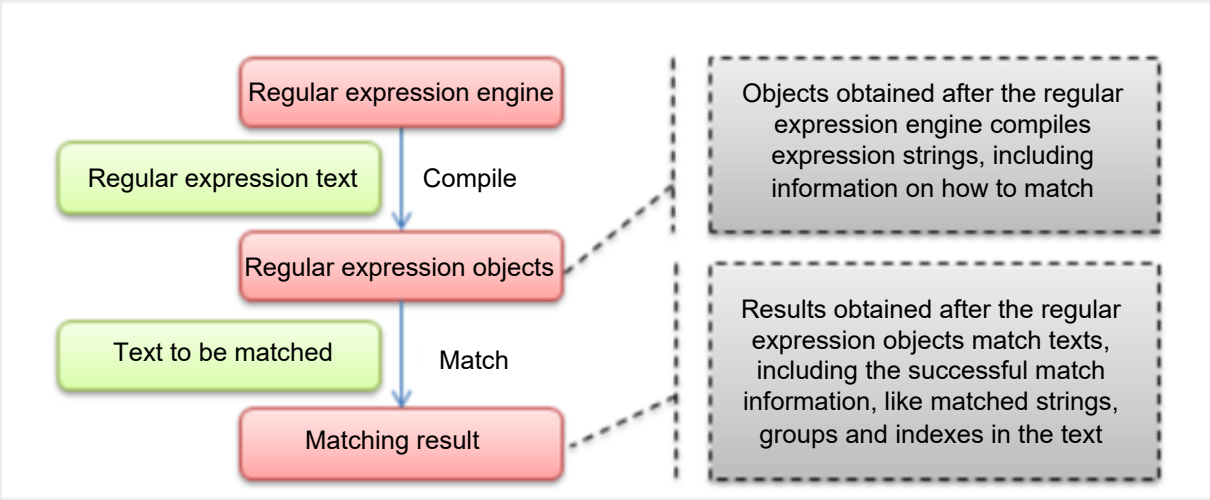
HUAWEI

Regular Expressions (1)

- A regular expression is a string of characters and special symbols that describe a pattern's repetition or multiple characters, and therefore a regular expression can match a series of strings with similar characteristics in a pattern.
- Regular expressions provide the basis for advanced text pattern matching, extraction, and/or text-style search and replace functions.
- Python supports regular expressions by using the re modules in the standard library.

Matching Process for Regular Expressions

- The approximate matching process for regular expressions is to match characters between regular expressions and texts. If each character matches, the match succeeds, and the match fails if there is any character match failure.



re Modules

- Python provides support for regular expressions by using the re module.
- The general step in using re is to compile the string form of a regular expression into a pattern instance, and then use the patterns instance to process the text and get the matching result (a match instance), and finally use the match instance to get the information and do other things.

```
import re

# Compile a regular expression into a patten object
pattern = re.compile(r'hello')

# Match text with pattern and return none if no match
match = pattern.match('hello world!')

if match:
    # Get group information using match
    print(match.group())

# Output
hello
```

re Module Functions and Regular Expression Object Methods

Function/Method	Description	Example	res.group()/res
compile(pattern,flag=0)	Compiles a regular expression pattern using any optional flag and returns regular expression objects.	res = re.compile(".*") print res.search("abcd").group()	abcd
match(pattern,string,flag=0)	Matches from the start of string.	res = re.match(".*","abcdxxxx")	abcd
search(pattern,string,flag=0)	Matches from any position of string.	res = re.search(".*",“xxxabcdxx")	abcd
findall(pattern,string,flag=0)	Finds all regular expression patterns in strings and returns a list.	res = re.findall("a", "abdadafaf")	['a','a','a','a']
finditer(pattern,string,flag=0)	Finds all regular expression patterns in strings and returns an iterator.	res = re.finditer("a", "abdadafaf") print res.next().group()	a
split(pattern,string,max=0)	Splits a string into a list by regular expression pattern.	re.split(",","li,yang,zhao")	['li','yang','zhao']
sub(pattern,repl,string,count=0)	Counts the positions with repl regular expressions in strings.	res = re.sub(",","-","l,y,z")	l-y-z

compile

- This method is the factory method of the pattern class, which is used to compile a regular expression in a string as a patterns object.

Patterns

- A pattern object is a compiled regular expression that can be matched to a search by a series of methods provided by pattern.

match

- The match object is a matching result that contains a lot of information about the match, and you can use the readable properties or methods provided by match to get that information.
- Match attributes:

Special Symbols and Characters - Symbols (1)

Symbol	Description	Matched Expression	res.group()
literal	Matches literal values of text strings.	res=re.search(“foo”,”xxxfooxxx”)	foo
re1|re 2	Matches regular expressions re1 or re2.	res=re.search(“foo|bar”,”xxxfooxxx”)	foo
.	Matches any character (except \n).	res=re.search(“b.b”,”xxxbobxxx”)	bob
^	Matches string start.	res=re.search("^b.b", "bobx xx")	bob
$	Matches string end.	res=re.search("b.b$","xx xbob")	bob
*	Matches regular expressions that appear none or many times (from string start).	res= re.search("bob*","bobbo") res1= re.search(".*","bobboddd")	Bobb bobboddd
+	Matches regular expressions that appear once or many times.	res= re.search("bob+","xxxxbobbbbob")	bobbbb

Special Symbols and Characters - Symbols (2)

Symbol	Description	Matched Expression	res.group()
?	Matches regular expressions that do not appear or appear once.	res=re.search("bob?","bobbod")	bob
{N}	Matches regular expressions that appear N times.	res=re.search("bob{2}","bobbbbod")	bobb
{M,N}	Matches regular expressions that appear M to N times.	res=re.search("bob{2,3}","bobbbbod")	bobbb

Special Symbols and Characters - Characters

Character	Description	Matched Expression	res.group()
\d	Any decimal number (\D does not match any decimal number).	res=re.search("xx\dxx","oxx4xxo")	xx4xx
\w	Matches any letter or number (\W means no match).	res=re.search("xx\w\wxx","oxxa4xxo")	xxa4xx
\s	Matches any space character (\S means no match).	res=re.search("xx\sxx","oxx xxo")	xx xx
\b	Matches any letter boundary (\B means reverse).	res=re.search(r"\bthe","xxx the xxx")	the
\N	Matches saved sub-group.		
\c	Matches any special character c one by one.	res=re.search("*","x*x")	*
\A(\Z)	Matches string start (or end).	res=re.search("\ADear","Dear Mr.Li")	Dear

re.match function

re.match tries to match a mode from the string start position. If no mode is matched from the string start, match() returns none.

```
re.match(pattern, string, flags=0)
Instance:
>>>import re
>>>print(re.match('www', 'www.runoob.com').span())    # Match at start
>>>print(re.match('com', 'www.runoob.com'))           # Match not at start
```

Output:

```
(0, 3)
None
```

re.search method

re.search scans the entire string and returns the first successful match.

```
re.search(pattern, string, flags=0)
```

Instance:

```
>>>import re
>>>line = "Cats are smarter than dogs"
>>>searchObj = re.search( r'(.*) are (.*?) .*', line, re.M|re.I)
>>>if searchObj:
>>>print("searchObj.group() : ", searchObj.group())
>>>print("searchObj.group(1) : ", searchObj.group(1))
>>>print("searchObj.group(2) : ", searchObj.group(2))
>>>else:
>>>print("Nothing found!!")
```

Execution result of the above instance:

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

Index and replace

The re module of Python provides re.sub to replace matched items in strings.

```
re.sub(pattern, repl, string, count=0, flags=0)


>>>import re
>>>phone = "2004-959-559 # This is an oversea telephone number"
# Delete Python comments in strings
>>>num = re.sub(r'#.*$', "", phone)
>>>print("The telephone number is", num)
# Delete non-number (-) strings
>>>num = re.sub(r'\D', "", phone)
>>>print("The telephone number is ", num)
```

Result:

```
The telephone number is: 2004-959-559
The telephone number is:    2004959559
```

re.compile function

The compile function compiles regular expressions and creates a regular expression (pattern) object, which will be used by the match() and search() functions.

```
re.compile(pattern[, flags])


>>>import re
>>> pattern = re.compile(r'\d+')                    # Match at least one number
>>> m = pattern.match('one12twothree34four')        # Search head, no match
>>>print(m)
None
>>> m = pattern.match('one12twothree34four', 2, 10) # Match from ‘e’, no match
>>>print(m)
None
```

re.compile function (Cont.)

```
>>> m = pattern.match('one12twothree34four', 3, 10) # Match from '1', matched
>>>print(m)                                          # Return a match object
<_sre.SRE_Match object at 0x10a42aac0>
>>> m.group(0)      # Ignorable 0
'12'
>>> m.start(0)      # Ignorable 0
3
>>> m.end(0)        # Ignorable 0
5
>>> m.span(0)       # Ignorable 0
(3, 5)
```

re.finditer

re.finditer finds all strings that match regular expressions, and returns them as an iterator.

```
re.finditer(pattern, string, flags=0)


>>>import re
>>>it = re.finditer(r"\d+","12a32bc43jf3")
>>>for match in it:
>>>print(match.group())
Output:
12
32
43
3
```

re.split

The split method splits matched strings and returns a list. For example:

```
>>>import re
>>> re.split('\W+', 'runoob, runoob, runoob.')
['runoob', 'runoob', 'runoob', '']
>>> re.split('(\W+)', ' runoob, runoob, runoob.')
['', ' ', 'runoob', ', ', 'runoob', ', ', 'runoob', '.', '']
>>> re.split('\W+', ' runoob, runoob, runoob.', 1)
['', 'runoob, runoob, runoob.']
>>> re.split('a+', 'hello world')      # Split will not split unmatched strings
['hello world']
```

Contents

Python File Manipulation

- File manipulation is of great importance to programming languages, and information technologies will be meaningless if data cannot be persistently read, saved, or used.
- Common types of file manipulation include opening and closing files, reading and writing files, and backing up files.

File Manipulation (1)

- Opening a file
  - f.open('file name','access mode')
  - Common access modes:

Access Mode	Description
r	Opens a file only for reading.
w	Opens a file only for writing.
a	Opens a file only for addition.
rb	Opens a file using the binary format only for reading.
wb	Opens a file using the binary format only for writing.
ab	Opens a file using the binary format only for addition.
r+	Opens a file for reading.
w+	Opens a file for writing.
a+	Opens a file for addition.
rb+	Opens a file using the binary format for reading.
wb+	Opens a file using the binary format for writing.
ab+	Opens a file using the binary format for addition.

File Manipulation (2)

- Writing data:

```
f = open(“name.txt”,“w”)
f.write(“libai”)
f.close()
```

- Reading data:

```
f = open(“name.txt”,“r”)

lines = f.realines()
for line in lines:
    print(line)
```

- Closing a file:

```
f.close()
```

Renaming Files in a Batch

- Get the target folder:
  - import os
  - dirName = input(“enter specified folder:”)
- Get names of files in the target folder:
  - fileNames = os.listdir(dirName)
  - os.chdir(dirName)
- Rename
  - for name in fileNames:
  - os.rename(name,“zhangsan”+name)

Other File Manipulation Functions

- Writing and reading files:
  - f.write("a") f.write (str) writes a string f.writeline (). f.readlines () is similar to the following read class.
  - f.read() reads all. f.read (size) reads characters of the size number from a file.
  - f.readline() reads a line, and returns an empty string to the file end. f.readlines() reads all, and returns a list. Each element of the list represents a row containing the " \n" \.
  - f.tell() returns the current file read location.
  - f.seek (off, where) locates the file read/write location. off represents an offset, a positive number means offset to the file end, and a negative number means offset to the file start.
  - where: 0 means counting from the beginning, 1 means counting from the current position, and 2 means counting from the end.
  - f.flush() flushes the cache.

Hands ON

- Get the Calendar month of your birth month in your birth year
- For the multiline string below:

  a. match with the phone numbers

  b. match with the website

```
S= """
abcde12378
www.myweb.com

342-555-6666
444.908.6849
"""
```

```
import calendar
cal = calendar.month(2012, 10)
print("output calendar of October 2012:")
print(cal)

import re
s= """
abcde12378
www.myweb.com

342-555-6666
444.908.6849
"""
pattern = re.compile(r"\d\d\d.\d\d\d.\d\d\d\d")
matches = pattern.finditer(s)
for match in matches:
    print(match)
web = re.search("www.*com", s)
print(web)
```

MCQ

1. Python is an object-oriented programming language. Which of the following are Python objects? ( )

A: function

B: module

C: number

D: string

ALL

2. Which of the following are not Python file object manipulations? ( )

A: open

B: delete

C: read

D: write

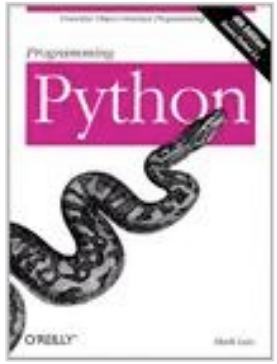
delete

Summary of the Chapter

- This course focuses on the Python language and its basic syntax, such as the Python compilation environment and installation of digital expressions, variables, statements, strings, access to user input, functions, modules and other common operations.

More Information

- Official site:
  - www.python.org
- References
  - Learning Python
  - Python Standard Library
  - Programming Python



Recommended for Learning

- Huawei e-learning site:
  - http://support.huawei.com/learning/Index!toTrainIndex
- Huawei knowledge base on the support website:
  - http://support.huawei.com/enterprise/servicecenter?lang=zh

Thanks

www.huawei.com