Programming in Python

GAVRILUT DRAGOS

COURSE 7

Time module

Implements function that allows one to work with time:

- Get current time
- Format time
- Sleep
- Time zone information

Details about **time** module in Python:

- Python 2: https://docs.python.org/2/library/time.html#module-time
- Python 3: https://docs.python.org/3/library/time.html#module-time

Time module

Usage:

Python 2.x / 3.x

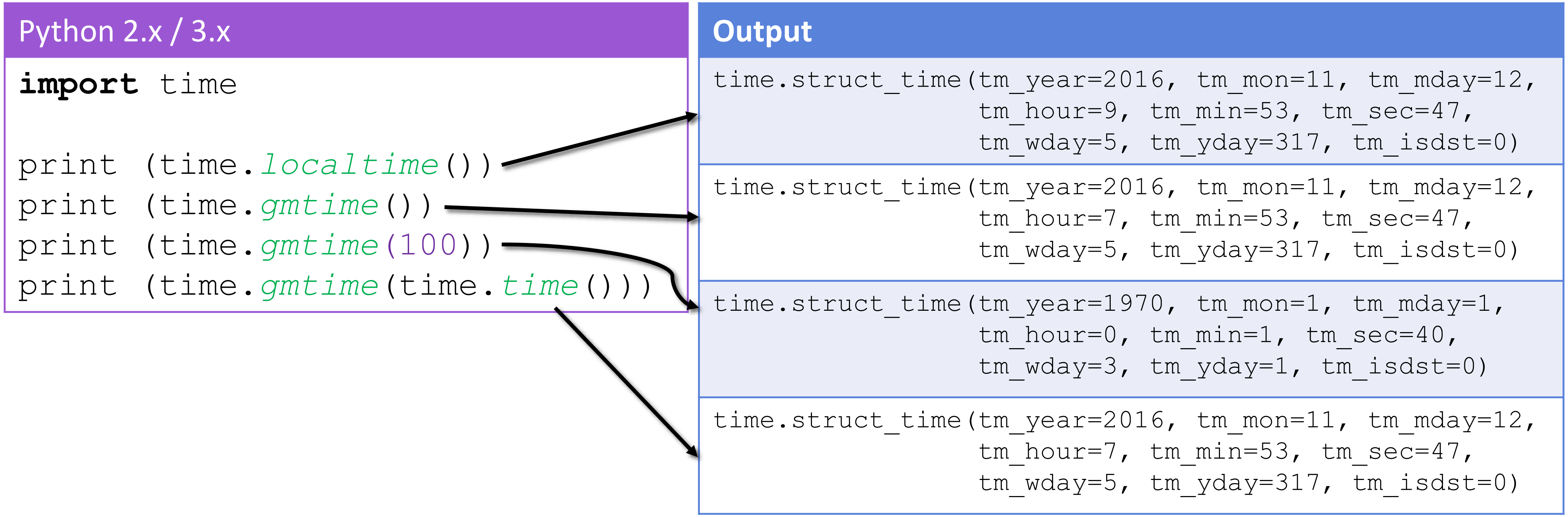
```
import time
w = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
print ("Time in seconds:",time.time())
print ("Today          :",time.ctime())
tmobj = time.localtime()
print ("Year           :",tmobj.tm_year)
print ("Month          :",tmobj.tm_mon)
print ("Day            :",tmobj.tm_mday)
print ("Day of week    :",w[tmobj.tm_wday])
print ("Day from year  :",tmobj.tm_yday)
print ("Hour           :",tmobj.tm_hour)
print ("Min            :",tmobj.tm_min)
print ("Sec            :",tmobj.tm_sec)
```

Output

```
Time in seconds: 1478936424.0649655
Today     : Sat Nov 12 09:40:24 2016
Year           : 2016
Month          : 11
Day            : 12
Day of week    : Saturday
Day from year  : 317
Hour           : 9
Min            : 40
Sec            : 24
```

Time module

Both **localtime** and **gmtime** have one parameter (the number of seconds from 1970). If this parameter is provided the time object will be the time computed based on that number. Otherwise the time object will be the time based on time.time () (current time) value.



Python 2.x / 3.x

```python
import time

print (time.localtime())
print (time.gmtime())
print (time.gmtime(100))
print (time.gmtime(time.time()))
```

Output

```
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=12,
                 tm_hour=9, tm_min=53, tm_sec=47,
                 tm_wday=5, tm_yday=317, tm_isdst=0)
```

```
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=12,
                 tm_hour=7, tm_min=53, tm_sec=47,
                 tm_wday=5, tm_yday=317, tm_isdst=0)
```

```
time.struct_time(tm_year=1970, tm_mon=1, tm_mday=1,
                 tm_hour=0, tm_min=1, tm_sec=40,
                 tm_wday=3, tm_yday=1, tm_isdst=0)
```

```
time.struct_time(tm_year=2016, tm_mon=11, tm_mday=12,
                 tm_hour=7, tm_min=53, tm_sec=47,
                 tm_wday=5, tm_yday=317, tm_isdst=0)
```

Time module

Use **strftime** to time object to a specified string representation:

Abreviation	Description
%H	Hour in 24 hour format
%I	Hour in 12 hour format
%Y	Year (4 digits)
%m	Month (decimal)
%B	Month (name)

Abreviation	Description
%M	Minute
%S	Seconds
%A	Day of week (name)
%d	Day of month (decimal)
%p	AM or PM

Python 2.x / 3.x

```
import time
tobj = time.localtime()
print (time.strftime("%H:%M:%S - %Y-%m-%d",tobj))
print (time.strftime("%I%p:%M:%S - %B",tobj))
print (time.strftime("%B,%A %d %Y",tobj))
```

Output

```
10:25:20 - 2016-11-12
10AM:25:20 - November
November,Saturday 12 2016
```

Time module

strftime if used without a time object applies the string format to the current time.

Time module also has a function **sleep** that receives one parameter (the number of seconds the current script has to wait until it continues its execution).

Python 2.x / 3.x

```
import time

for i in range(0,8):
        print (time.strftime("%H:%M:%S"))
        time.sleep(2)#sleep 2 seconds
```

If no time object structure is provided, **strftime** function will use current time to apply the specified format.

Output

```
11:46:15
11:46:17
11:46:19
11:46:21
11:46:23
11:46:25
11:46:27
11:46:29
```

hashlib module

Implements function that allows one to compute different cryptographic functions:

- MD5
- SHA-1
- SHA-224
- SHA-384
- SHA-512

Details about **hashlib** module in Python:

- Python 2: https://docs.python.org/2/library/hashlib.html
- Python 3: https://docs.python.org/3/library/hashlib.html

hashlib module

Each hashlib object has an **update** function (to update the value of the hash) and a **digest** or **hexdigest** function(s) to compute the final hash.

Python 2.x / 3.x

```
import hashlib

m = hashlib.md5()
m.update(b"Today")
m.update(b" I'm having")
m.update(b" a Python ")
m.update(b"course")
print (m.hexdigest())
```

Output

```
9dea650a4eab481ec0f4b5ba28e3e0b8
```

```
import hashlib

print (hashlib.md5(b"Today I'm having a Python course").hexdigest())
```

hashlib module

Each hashlib object has an **update** function (to update the value of the hash) and a **digest** or **hexdigest** function(s) to compute the final hash.

Python 2.x / 3.x

```
import hashlib

m = hashlib.md5()
m.update(b"Today")
m.update(b" I'm having")
m.update(b" a Python ")
m.update(b"course")
print (m.hexdigest())
```

```
import hashlib

print (hashlib.md5(b"Today I'm having a Python course").hexdigest())
```

The **<b>** prefix in front of a string is ignored in Python 2. In Python 3 means that the string is a byte list. **update** method requires a list of bytes (not a string). However, in Python 2 it can be used without the prefix <**b**>

hashlib module

Hashes are often use on files (to associate the content of a file to a specific hash).

Python 2.x / 3.x

```
import hashlib

def GetFileSHA1(filePath):
    m = hashlib.sha1()
    m.update(open(filePath,"rb").read())
    return m.hexdigest()


print (GetFileSHA1("< a file path >"))
```

Output

```
cad7a796be26149218a76661d316685d7de2d56d
```

While this example is ok, keep in mind that it loads the entire file content in memory !!!

hashlib module

The correct way to do this (having a support for large files is as follows):

Python 2.x / 3.x

```
import hashlib
def GetFileSHA1(filePath):
    try:
        m = hashlib.sha1()
        f = open(filePath,"rb")
        while True:
            data = f.read(4096)
            if len(data)==0: break
            m.update(data)
        f.close()
        return m.hexdigest()
    except:
        return ""
```

JSON module

Python has several implementations for data serialization.

- JSON
- Pickle
- Marshal

Documentation for JSON:

- Python 2: https://docs.python.org/2/library/json.html
- Python 3: https://docs.python.org/3/library/json.html

JSON module

JSON functions:

- **json.dump** (obj, fp, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *cls=None*, *indent=None*, *separators=None, default=None, sort_keys=False, **kw*)
- **json.dumps**(obj, *skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*) ➔ to obtain the string representation of the obj in JSON format
- **json.load**(fp, *cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw*)
- **json.loads**(s, *encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw*)

JSON module

Usage (serialization):

Python 2.x / 3.x

```
import json

d = { "a":[1,2,3],
      "b":100,
      "c":True
}
s = json.dumps(d)
open("serialization.json","wt").write(s)
print (s)
```

serialization.json

```
FileAddr     000 001 002 003 004 005 006 007     Text
000000000    123 034 099 034|058 032 116 114     {"c": tr
000000008    117 101 044 032|034 097 034 058     ue, "a":
000000016    032 091 049 044|032 050 044 032      [1, 2,
000000024    051 093 044 032|034 098 034 058     3], "b":
000000032    032 049 048 048|125                  100}
```

Output

```
{"a": [1, 2, 3], "b": 100, "c": true}
```

JSON module

Usage (de-serialization):

Python 2.x / 3.x

```
import json

data = open("serialization.json","rt").read()
d = json.loads(data)
print (d)
```

```
import json

d = json.load(open("serialization.json","rt"))
print (d)
```

Output

```
{"a": [1, 2, 3], "b": 100, "c": true}
```

PICKLE module

Pickle is another way to serialize objects in Python. The serialization is done in a binary mode.

Pickle can also serialize:

- Functions (defined using **def** and not lambda)
- classes
- Functions from modules

Documentation for PICKLE :

- Python 2: https://docs.python.org/2/library/pickle.html
- Python 3: https://docs.python.org/3/library/pickle.html

PICKLE module

PICKLE functions:

- **pickle.dump** (obj, file, *protocol=None, *, fix_imports=True*)
- **pickle.dumps**(obj, *protocol=None, *, fix_imports=True*) ➔ to obtain the buffer representation of the obj in pickle format
- **pickle.load**(file, **, fix_imports=True, encoding="ASCII", errors="strict"*)
- **pickle.loads**(byte_object, **, fix_imports=True, encoding="ASCII", errors="strict"*)

PICKLE support multiple version. Be careful when you serialize with Python 2 and try to de-serialize with Python 3 (not all version supported by Python 3 are also supported by Python 2).

If you are planning to switch between versions, either check pickle.HIGHEST_PROTOCOL to see if the hightest protocol are compatible, or use **0** as the protocol value.

PICKLE module

Usage (serialization):

Python 2.x / 3.x

```
import pickle

d = {
    "a":[1,2,3],
    "b":100,
    "c":True
}
```

serialization.json

```
FileAddr      000 001 002 003 004 005 006 007       Text
000000000     128 003 125 113|000 040 088 001       Ç♥}q (X☺
000000008     000 000 000 099|113 001 136 088         cq☺êX
000000016     001 000 000 000|097 113 002 093       ☺    aq☻]
000000024     113 003 040 075|001 075 002 075       q♥(K☺K☻K
000000032     003 101 088 001|000 000 000 098       ♥eX☺    b
000000040     113 004 075 100|117 046               q♦Kdu.
```

```
buffer = pickle.dumps(d) #buffer = pickle.dumps(d,0) (safety)
open("serialization.pickle","wb").write(buffer)
```

Pickle need a file to be open in binary mode !

PICKLE module

Usage (de-serialization):

Python 2.x / 3.x

```
import pickle

data = open("serialization.pickle","rb").read()
d = pickle.loads(data)
print (d)
```

```
import pickle

d = pickle.load(open("serialization.pickle","rb"))
print (d)
```

Output

```
{"a": [1, 2, 3], "b": 100, "c": true}
```

Marshal module

Marshal is another way to serialize objects in Python. The serialization is done in a binary mode. Designed for python compiled code (pyc). The binary result is platform-dependent !!!

Marshal functions:

- **marshal.dump** (value, file, *[version]*) marshal
- **marshal.dumps(**value, *[version]*) ➔ to obtain the binary representation of the obj in marshal format
- **marshal.load(**file)
- **marshal.loads(**string/buffer)

Documentation for Marshal :

- Python 2: https://docs.python.org/2/library/marshal.html#module-marshal
- Python 3: https://docs.python.org/3/library/marshal.html#module-marshal

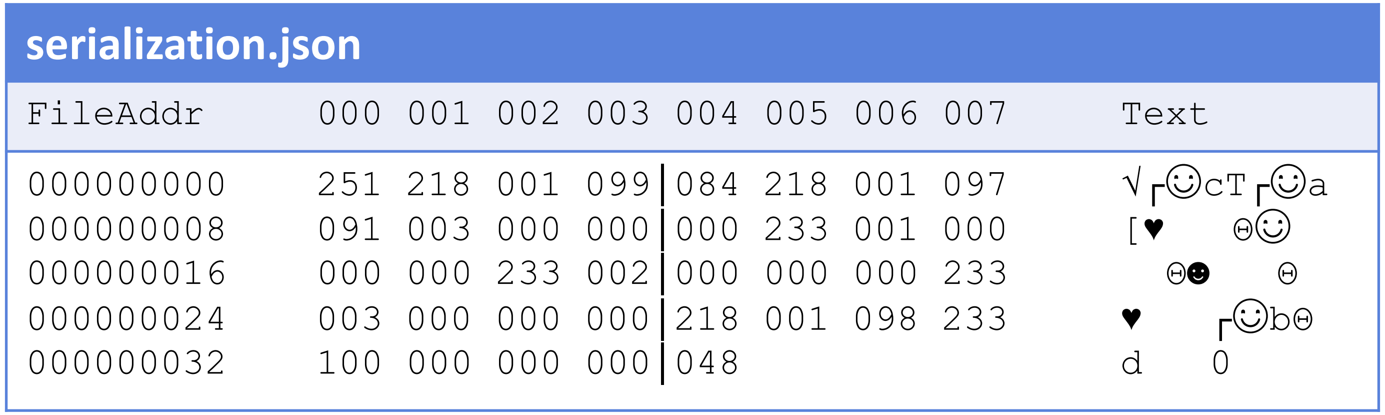
Marshal module

Usage (serialization):

Python 2.x / 3.x

```
import marshal

d = {
    "a":[1,2,3],
    "b":100,
    "c":True
}
```



serialization.json

```
FileAddr    000 001 002 003 004 005 006 007    Text

000000000   251 218 001 099|084 218 001 097    √┌☺cT┌☺a
000000008   091 003 000 000|000 233 001 000    [♥   Θ☺
000000016   000 000 233 002|000 000 000 233      Θ☻   Θ
000000024   003 000 000 000|218 001 098 233    ♥   ┌☺bΘ
000000032   100 000 000 000|048                d   0
```

```
buffer = marshal.dumps(d)
open("serialization.marshal","wb").write(buffer)
```

Marshal module

Usage (de-serialization):

Python 2.x / 3.x

```
import marshal

data = open("serialization.marshal","rb").read()
d = marshal.loads(data)
print (d)
```

```
import marshal

d = marshal.load(open("serialization.marshal","rb"))
print (d)
```

Marshal serialization has a different format in Python 2 and Python 3 (these two are not compatible).

Output

```
{"a": [1, 2, 3], "b": 100, "c": true}
```

Random module

Implements different random base functions:

- random.random() ➔ a random float number between 0 and 1
- random.randint(min,max) ➔ a random integer number between [min … max]
- random.choice(list) ➔ selects a random element from a list
- random.shuffle(list) ➔ shuffles the list
- random.sample(list,count) ➔ creates another list from the current one containing **count** elements

Details about **random** module in Python:

- Python 2: https://docs.python.org/2/library/random.html
- Python 3: https://docs.python.org/3/library/random.html

Random module

Usage:

Python 2.x / 3.x

```
import random

print (random.random())
print (random.randint(5,10))

l = [2,3,5,7,11,13,17,19]
print (random.choice(l))
print (random.sample(l,3))

random.shuffle(l)
print (l)
```

Output

```
0.9410874890940395
9
5
[19, 17, 11]
[13, 17, 11, 5, 2, 19, 7, 3]
```

ZipFile module

Implements different functions to work with a zip archive:

- List all elements from a zip archive
- Extract files
- Add files to archive
- Get file information
- etc

Details about **zipfile** module in Python:

- Python 2: https://docs.python.org/2/library/zipfile.html
- Python 3: https://docs.python.org/3/library/zipfile.html

ZipFile module

Listing the content of a zip archive:

Python 2.x / 3.x

```
import zipfile

z = zipfile.ZipFile("archive.zip")
for i in z.infolist():
    print (i.filename,
           i.file_size,
           i.compress_size)
z.close()
```

Output

```
MathOps/ 0 0
MathOps/Complex/ 0 0
MathOps/Complex/Series.py 117 79
MathOps/Complex/__init__.py 38 38
MathOps/Simple/ 0 0
MathOps/Simple/Arithmetic.py 54 52
MathOps/Simple/Bits.py 60 55
MathOps/Simple/__init__.py 87 84
MathOps/__init__.py 30 30
a.py 43 43
all.csv 62330588 8176706
```

ZipFile module

To extract a file from an archive:

Python 2.x / 3.x

```
import zipfile
z = zipfile.ZipFile("archive.zip")
z.extract("MathOps/Simple/Arithmetic.py","MyFolder")
z.close()
```

Arithmetic.py will be extracted to “MyFolder/MathOps/Simple/Arithmetic.py”

To extract all files:

Python 2.x / 3.x

```
import zipfile
z = zipfile.ZipFile("archive.zip")
z.extractall("MyFolder")
z.close()
```

ZipFile module

A file can also be opened directly from an archive. This is usually required if one wants to extract the content somewhere else or if the content needs to be analyzed in memory.

Python 2.x / 3.x

```
import zipfile

z = zipfile.ZipFile("archive.zip")
f = z.open("MathOps/Simple/Arithmetic.py")
data = f.read()
f.close()
open("my_ar.py","wb").write(data)
z.close()
```

Method **open** from zipfile returns a file-like object. You can also specify a password:
Format: ZipFile.*open*(name, *mode='r', pwd=None*)

ZipFile module

The following script creates a zip archive and add files to it:

Python 2.x / 3.x

```
import zipfile

z = zipfile.ZipFile("new_archive.zip","w",zipfile.ZIP_DEFLATED)
z.writestr("test.txt","some texts ...")
z.write("serialization.json")
z.write("serialization.json", "/dir/a.json")
z.writestr("/dir/a.txt","another text ...")
z.close()
```

writestr method writes the content of a string into a zip file.
write methods add a file to the archive.

When creating an archive one can specify a desire compression: ZIP_DEFLATE, ZIP_STORED, ZIP_BZIP2 or ZIP_LZMA.