THOMAS MORE

KEEPING TIME, SCHEDULING TASKS, AND LAUNCHING PROGRAMS

Joris Van Acoleyen

WAT?

- Time module
- Rounding numbers
- Datetime module
- Review of python’s time functions
- Multithreading
- Launching other programs from python
- Summary
- Questions
- Labo

TIME MODULE

- De tijd module laat toe huidige tijd te lezen
- De belangrijkste functies zijn time.time() en time.sleep()
- Om hiermee te werken moet deze ook geïmporteerd worden met: import time

TIME MODULE: TIME.TIME()

- Unix epoch (January 1, 1970, at midnight, UTC) is een vaak gebruikte tijdsreferentie bij programmeren
- Time.time() geeft een float waarde

```
>>> import time
>>> time.time()
1513195452.9234278
```

TIME MODULE: TIME.SLEEP()

- Kan een programma voor een beplaade tijd pauseren

```
>>> import time
>>> time.sleep(aantal
seconden)
```

ROUNDING NUMBERS

- Herhaling
- Round(var of float, aantal cijfers)

```
>>> import time
>>> now = time.time()
>>> now
1425064108.017826
>>> round(now, 2)
```

DATETIME MODULE

- Beter leesbare variant van time.time()
- Datetimeformat (yyyy,mm,dd,h,m,s,µs)
- Heeft ook atributen om alleen dag of maand weertegeven

```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2017, 12, 13, 21, 38, 38, 320166)
>>> datetime.datetime.now().month
12
```

DATETIME MODULE

- Een unix epoch timestamp kan ook geconverteerd worden
- 100000000 seconden na unix epoch

```
>>> datetime.datetime.fromtimestamp(100000000)
datetime.datetime(1973, 3, 3, 10, 46, 40)
```

```
>>> import time
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2017, 12, 13, 21, 53, 15, 73119)
```

DATETIME MODULE

- Er kan ook worden vergeleken met datetime en operands

```
halloween2016 = datetime.datetime(2017, 10, 31, 0, 0, 0)
newyears2018 = datetime.datetime(2018, 1, 1, 0, 0, 0)
oct31_2017 = datetime.datetime(2017, 10, 31, 0, 0, 0)

>>> halloween2017 == oct31_2017
True
>>> halloween2015 > newyears2016
False
>>> newyears2016 > halloween2015
True
>>> newyears2016 != oct31_2015
True
```

DATETIME MODULE

- Timedelta een duur van tijd ipv een moment in tijd

```
delta = datetime.timedelta(days=11, hours=10,
minutes=9, seconds=8)
>>> delta.days, delta.seconds, delta.microseconds
(11, 36548, 0)
>>> delta.total_seconds()
986948.0
>>> str(delta)
'11 days, 10:09:08'
```

DATETIME MODULE

vb.

```
dt = datetime.datetime.now()
>>> dt
datetime.datetime(2015, 2, 27, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2017, 11, 23, 18, 38, 50, 636181)
```

Er kan ook bij een datetime een timedelta worden bijgeteld, afgetroken, …

DATETIME MODULE

- Pauzeren to bepaalde datum

```
import datetime
import time
halloween2017 = datetime.datetime(2017, 10, 31, 0, 0, 0)
while datetime.datetime.now() < halloween2017:
time.sleep(1)
```

DATETIME MODULE

- Datetime naar string

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2015/10/21 16:29:00'
```

strftime directive	Meaning
%Y	Year with century, as in '2014'
%y	Year without century, '00' to '99' (1970 to 2069)
%m	Month as a decimal number, '01' to '12'
%B	Full month name, as in 'November'
%b	Abbreviated month name, as in 'Nov'
%d	Day of the month, '01' to '31'
%j	Day of the year, '001' to '366'
%w	Day of the week, '0' (Sunday) to '6' (Saturday)
%A	Full weekday name, as in 'Monday'
%a	Abbreviated weekday name, as in 'Mon'
%H	Hour (24-hour clock), '00' to '23'
%I	Hour (12-hour clock), '01' to '12'
%M	Minute, '00' to '59'
%S	Second, '00' to '59'
%p	'AM' or 'PM'
%%	Literal '%' character

DATETIME MODULE

- String naar datetime

```
>>> datetime.datetime.strptime('October 21, 2015', '%B %d, %Y')
datetime.datetime(2015, 10, 21, 0, 0)
>>> datetime.datetime.strptime('2015/10/21 16:29:00', '%Y/%m/%d
%H:%M:%S')
datetime.datetime(2015, 10, 21, 16, 29)
>>> datetime.datetime.strptime("October of '15", "%B of '%y")
datetime.datetime(2015, 10, 1, 0, 0)
>>> datetime.datetime.strptime("November of '63", "%B of '%y")
datetime.datetime(2063, 11, 1, 0, 0)
```

REVIEW OF PYTHON'S TIME FUNCTIONS

- A Unix epoch timestamp (used by the time module) is a float or integer value of the number of seconds since 12 am on January 1, 1970, UTC.
- A datetime object (of the datetime module) has integers stored in the attributes year, month, day, hour, minute, and second.
- A timedelta object (of the datetime module) represents a time duration, rather than a specific moment.
- The time.time() function returns an epoch timestamp float value of the current moment.
- The time.sleep(*seconds*) function stops the program for the amount of seconds specified by the *seconds* argument.
- The datetime.datetime(*year*, *month*, *day*, *hour*, *minute*, *second*) function returns a datetime object of the moment specified by the arguments. If *hour*, *minute*, or *second* arguments are not provided, they default to 0.
- The datetime.datetime.now() function returns a datetime object of the current moment.
- The datetime.datetime.fromtimestamp(*epoch*) function returns a datetime object of the moment represented by the *epoch* timestamp argument.
- The datetime.timedelta(*weeks*, *days*, *hours*, *minutes*, *seconds*, *milliseconds*, *microseconds*) function returns a timedelta object representing a duration of time. The function's keyword arguments are all optional and do not include *month* or *year*.
- The total_seconds() method for timedelta objects returns the number of seconds the timedelta object represents.
- The strftime(*format*) method returns a string of the time represented by the datetime object in a custom format that's based on the *format* string.
- The datetime.datetime.strptime(*time_string*, *format*) function returns a datetime object of the moment specified by *time_string*, parsed using the *format* string argument. See Table 15-1 for the format details.

MULTITHREADING

- Zorgt ervoor da er meerdere programma's tegelijk kunnen draaien

Vb. single thread

```
import time, datetime
startTime = datetime.datetime(2029, 10, 31, 0, 0, 0)
while datetime.datetime.now() < startTime:
time.sleep(1)
print('Program now starting on Halloween 2029')
```

MULTITHREADING

- Vb. multithread

```
import threading, time
print('Start of program.')
def takeANap():
        time.sleep(5)
        print('Wake up!')
threadObj = threading.Thread(target=takeANap)
threadObj.start()
print('End of program.')
```

Niet wachten op time.sleep() om te printen

MULTITHREADING

- Passing arguments to thread's target function
- Kan voor problemen zorgen met variabelen

```
>>> print('Cats', 'Dogs', 'Frogs', sep=' & ')
Cats & Dogs & Frogs

>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats',
'Dogs', 'Frogs'],
kwargs={'sep': ' & '})
>>> threadObj.start()
Cats & Dogs & Frogs
```

LAUNCHING OTHER PROGRAMS FROM PYTHON

- Windows:

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
```

- Linux:

```
>>> import subprocess
>>> subprocess.Popen('/usr/bin/gnome-calculator')
```

LAUNCHING OTHER PROGRAMS FROM PYTHON

- Poll(): kijkt na of het proces nog actief is
- Wait(): wacht tot het proces gedaan is

```
>>> calcProc =
subprocess.Popen('c:\\Windows\\System32\\calc.exe')
>>> calcProc.poll() == None
True
>>> calcProc.wait()
0
>>> calcProc.poll()
0
```

LAUNCHING OTHER PROGRAMS FROM PYTHON

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe',
'C:\\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>

>>> subprocess.Popen(['C:\\python34\\python.exe',
'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

OS X:

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

SUMMARY

- The Unix epoch
- Time.time()
- Time.sleep()
- Datetime
- Multithreading
- Opening programs with python

QUESTIONS

1. What is the Unix epoch?
2. What function returns the number of seconds since the Unix epoch?
3. How can you pause your program for exactly 5 seconds?
4. What does the round() function return?
5. What is the difference between a datetime object and a timedelta object?
6. Say you have a function named spam(). How can you call this function and run the code inside it in a separate thread?
7. What should you do to avoid concurrency issues with multiple threads?
8. How can you have your Python program run the *calc.exe* program located in the *C:\Windows\System32* folder?

LABO

- Trac