Python Batteries - oTree Concepts - Tutorial

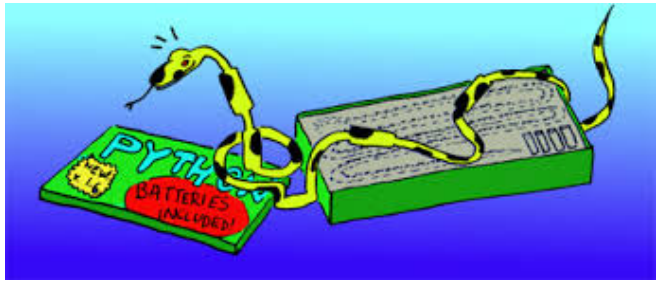
Juan Cabral - jbc.develop@gmail.com

Jan, 2018

Python Batteries

- The Python source distribution has long maintained the philosophy of **"batteries included"** – having a rich and versatile standard library which is immediately available, without making the user download separate packages. This gives the Python language a head start in many projects.
- **However, the standard library modules aren't always the best choices for a job.**
- How many functionalities have the python batteries? Try:

```
import antigravity
```



Python Batteries - os module

This module provides a portable way of using operating system dependent functionality.

- The design of all built-in operating system dependent modules of Python is such that as long as the same functionality is available, it uses the same interface.
- Extensions peculiar to a particular operating system are also available through the os module, but using them is of course a threat to portability.
- If not separately noted, all functions that claim "Availability: Unix" are supported on Mac OS X, which builds on a Unix core.
- Docs: https://docs.python.org/3/library/os.html

os module - Basic operations

- operative system family

```
# check platform module for more details
>>> os.name
'posix'
```

- The current directory

```
>>> os.getcwd()
'/home/jbcabral/projects/otree_bogota2018/src'
```

os module - Basic operations

- create a new directory with the name “name”

```
>>> os.mkdir("name")
```

- create directory `name` and their parent `parent`

```
>>> os.makedirs("parent/name")
```

- remove file (also check the module `shutil`)

```
>>> os.remove("filename")
```

os module - Waliking though files

- List a directory

```
>>> os.path.listdir("path")
```

- Recursive listing

```
>>> for root, dnames, fnames in os.walk(path):
        for fname in fnames:
            print(os.path.join(root, fname))
```

os module - Paths

- Check if a path is a file

```
>>> os.path.isfile("/home/juan/.bashrc")
True
```

- Check if a path is a directory

```
>>> os.path.isdir("/home/juan/.bashrc")
False
```

- Check if a path exists

```
>>> os.path.exists("/home/juan/.bashrc")
True
```

os module - Path of the current module

```
PATH = os.path.abspath(os.path.dirname(__file__))
```

- Every module has a attribute called `__file__` with the relative path of the current file.
- `os.path.dirname` remove the "file" part of the `__file__` attribute.
- `os.path.abspath` convert the path to an absolute path.

Python Batteries - `datetime` module

The datetime module supplies classes for manipulating dates and times in both simple and complex ways. While date and time arithmetic is supported, the focus of the implementation is on efficient attribute extraction for output formatting and manipulation.

- Full Documentation: https://docs.python.org/3/library/datetime.html

datetime module: Basic

```
import datetime
now = datetime.datetime(2003, 8, 4, 12, 30, 45)

print(now)
# => 2003-08-04 12:30:45

print repr(now)
# => datetime.datetime(2003, 8, 4, 12, 30, 45)

print type(now)
# => <type 'datetime.datetime'>

print(now.year, now.month, now.day)
# => 2003 8 4

print(now.hour, now.minute, now.second)
# => 12 30 45
```

datetime module: Convert from another type

```
import datetime
import time

print(datetime.datetime(2003, 8, 4, 21, 41, 43))
# => 2003-08-04 21:41:43

datetime.datetime.today()

datetime.datetime.now()

datetime.datetime.fromtimestamp(time.time())

datetime.datetime.utcnow()

datetime.datetime.utcfromtimestamp(time.time())
```

datetime module: to String

```
>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2017, 12, 5, 23, 37, 53, 112972)

>>> now.ctime()
'Tue Dec  5 23:37:53 2017'

>>> now.isoformat()
'2017-12-05T23:37:53.112972'

>>> now.strftime("%Y%m%dT%H%M%S")
'20171205T233753'
```

datetime module: Algebra

```
>>> past = datetime.datetime.now()
>>> past
datetime.datetime(2017, 12, 5, 23, 37, 53, 112972)


>>> datetime.datetime.now() - past
datetime.timedelta(0, 26, 329369)

>>> datetime.datetime.now() - past
datetime.timedelta(0, 27, 864966)

>>> datetime.datetime.now() - past
datetime.timedelta(0, 29, 289356)
```

datetime module: Algebra 2

```
>>> past = datetime.datetime.now()
>>> past
datetime.datetime(2017, 12, 5, 23, 42, 54, 209964)


>>> delta = datetime.datetime.now() - past
datetime.timedelta(0, 24, 168670)

>>> now = datetime.datetime.now()
>>> now
datetime.datetime(2017, 12, 5, 23, 43, 55, 510720)

>>> now + delta
datetime.datetime(2017, 12, 5, 23, 44, 19, 679390)
```

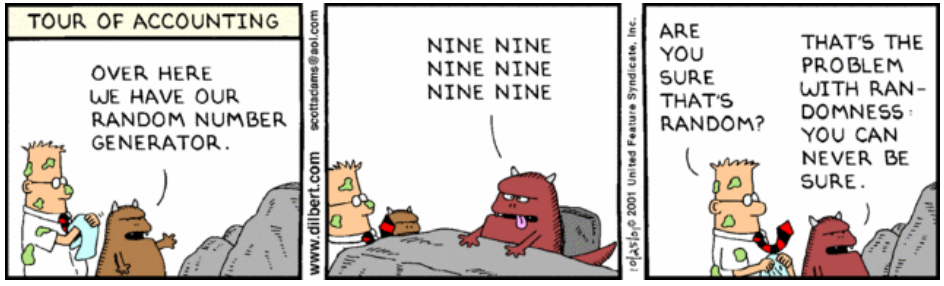
`datetime` module

Resume:

- The `datetime.datetime` type represents a date and a time during that day.
- The `datetime.date` type represents just a date, between year 1 and 9999
- The `datetime.time` type represents a time, independent of the date.
- The `datetime.timedelta` type represents the difference between two time or date objects.
- The `datetime.tzinfo` type is used to implement timezone support for time and datetime objects (this will not be cover in this tutorial).
- In servers always use **utcnow()**

Python Batteries - `random` module

- This module implements pseudo-random number generators for various distributions.
  For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.
  There are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions.
- Full Documentation:
  https://docs.python.org/3/library/random.html



random module: basic

```
>>> import random

# Random float:  0.0 <= x < 1.0
>>> random.random()
0.37444887175646646

# Random float:  2.5 <= x < 10.0
>>> random.uniform(2.5, 10.0)
3.1800146073117523

# Interval between arrivals averaging 5 seconds
>>> random.expovariate(1 / 5)
5.148957571865031

# Integer from 0 to 9 inclusive
>>> random.randrange(10)
7
```

random module: basic 2

```
# Even integer from 0 to 100 inclusive
>>> random.randrange(0, 101, 2)
26

# Single random element
>>> random.choice(['win', 'lose', 'draw'])
'draw'

# Shuffle a list
>>> deck = 'ace two three four'.split()
>>> random.shuffle(deck)
>>> deck
['four', 'two', 'ace', 'three']

# Four samples without replacement
>>> random.sample([10, 20, 30, 40, 50], k=4)
[40, 10, 50, 30]
```

random module: Simulations

```
>>> import collections
>>> import random

# Six roulette wheel spins
# (weighted sampling with replacement)
>>> random.choices(['red', 'black', 'green'],
                   [18, 18, 2], k=6)
['red', 'green', 'black', 'black', 'red', 'black']

# Deal 20 cards without replacement from a deck of
# 52 playing cards and determine the proportion of
# cards with a ten-value (a ten, jack, queen, or king).
>>> deck = collections.Counter(tens=16, low_cards=36)
>>> seen = random.sample(list(deck.elements()), k=20)
>>> seen.count('tens') / 20
0.15
```

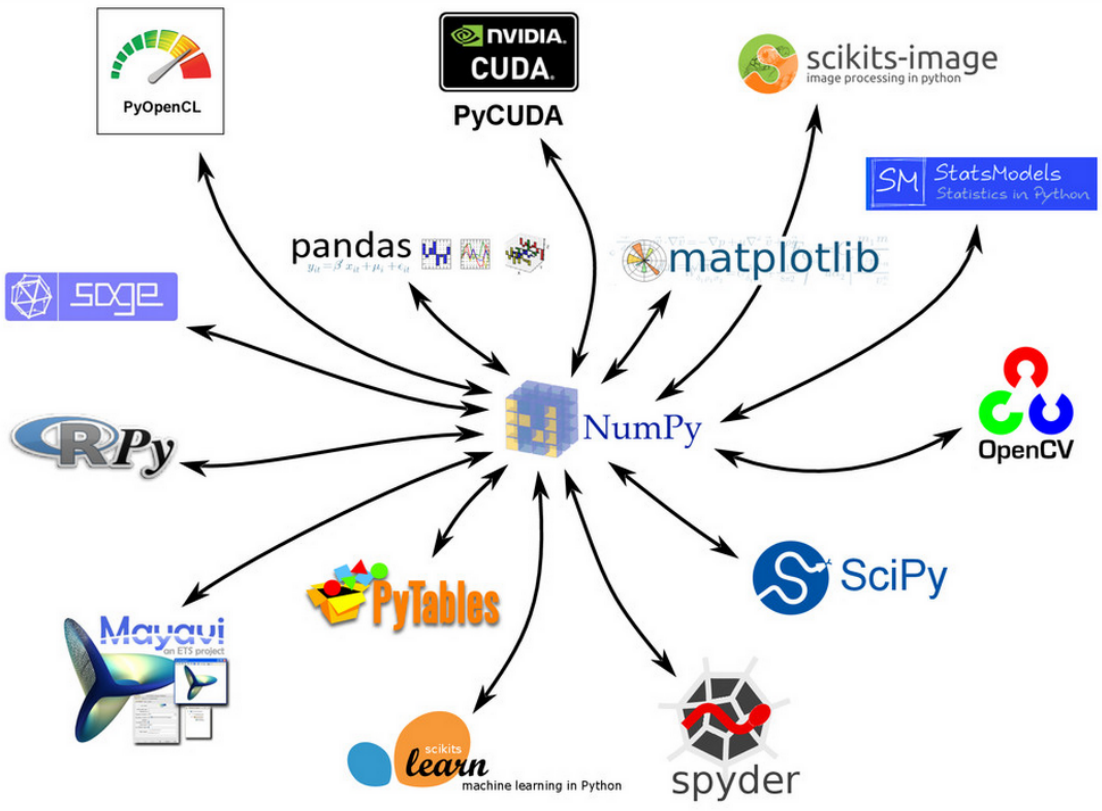
random module: Simulations 2

```
>>> import random
>>> import statistics

# Estimate the probability of getting 5 or more heads
# from 7 spins of a biased coin that settles on
# heads 60% of the time.
>>> def trial():
        return random.sample.choices(
              'HT', cum_weights=(0.60, 1.00),
         k=7).count('H') >= 5
>>> statistics.mean(trial() for _ in range(10000))
0.4169
```

`random` module: resume

- `random` and `statistics` are useful in many cases but is not feature complete.
- In complex cases you must install numpy and scipy



Python Batteries - `itertools` module

- This module implements a number of iterator building blocks
- Some provide streams of infinite length, so they should only be accessed by functions or loops that truncate the stream.
- Full Documentation: https://docs.python.org/3/library/itertools.html

Python Batteries - `itertools` module

- Repeat something n times

```
>>> for e in itertools.repeat(10, 3):
...     print(e)
10
10
10
```

- Repeat any iterable forever

```
>>> list_cycle = itertools.cycle([1,2,3])
>>> next(list_cycle)
1
>>> next(list_cycle)
2
>>> next(list_cycle)
3
>>> next(list_cycle)
1
```

Python Batteries - `itertools` module

- Simple filter

```
>>> for e in itertools.compress('ABCDE', [1,0,1,0,1]):
...     prin(e)
A
C
E
```

- Iterable concatenation

```
>>> for e in itertools.chain('A', 'DEF'):
...     print(e)
A
D
E
F
```

Python Batteries - `itertools` module

Combinatoric generators:

Iterator	Arguments	Results
`product()`	p, q, … [repeat=1]	cartesian product, equivalent to a nested for-loop
`permutations()`	p[, r]	r-length tuples, all possible orderings, no repeated elements
`combinations()`	p, r	r-length tuples, in sorted order, no repeated elements
`combinations_with_replacement()`	p, r	r-length tuples, in sorted order, with repeated elements
`product('ABCD', repeat=2)`		`AA AB AC AD BA BB BC BD CA CB CC CD DA DB DC DD`
`permutations('ABCD', 2)`		`AB AC AD BA BC BD CA CB CD DA DB DC`
`combinations('ABCD', 2)`		`AB AC AD BC BD CD`
`combinations_with_replacement('ABCD', 2)`		`AA AB AC AD BB BC BD CC CD DD`

References

- https://docs.python.org/3/library/shutil.html
- https://docs.python.org/3/library/statistics.html
- https://docs.python.org/3/library/collections.html
- https://docs.python.org/3/library/itertools.html
- https://docs.python.org/3/library/os.html
- https://docs.python.org/3/library/random.html
- https://docs.python.org/3/library/time.html
- https://docs.python.org/3/library/datetime.html
- https://www.python.org/dev/peps/pep-0206/
- http://otree.readthedocs.io/en/latest/