Unit 2: Python extras

Here we give some Python extras and examples on

- Generators
- Lists
- List comprehension
- Profiling
- Documentation
- Class methods
2. 1 Generators

Definition 3. A generator generates objects (to be passed to for loop). Similar to a list except that the objects need not to exist before entering the loop.

A generator-like object is range: (xrange in Python 2.7)

```python
for i in range(100000000):
    if i > 10:
        break
```

See also

```python
rr = range(20)
print(rr)
```

rr is not a list, but a tool which can be used to generate a list:

```python
rll = list(rr)
```
2. 2 Generators: Python definition

Creation of generators is possible with the keyword `yield`:

```python
def odd_numbers(n):
    "generator for odd numbers less than n"
    for k in range(n):
        if k % 2 == 1:
            yield k
```

Then you call it as:

```python
g = odd_numbers(10)
for k in g:
    ... # do something with k
```
2.3 Infinite Generators

Note that, just as in mathematics, generators may be infinite:

```python
def odd_numbers():
    "generator for all odd numbers"
    k=1
    while True:
        if k % 2 == 1:
            yield k
        k+=1

on=odd_numbers()
print(on.__next__())
```

Note: Finite generator objects are exhausted after their use.
2.4 Generator tools

- **enumerate** is used to enumerate another generator:

```python
g = odd_numbers(10)
for i, x in enumerate(g):
    print(i, x, end='; ')
# result: 0 1 ; 1 3 ; 2 5 ; 3 7 ; 4 9 ;
```

- **reversed** creates a generator from a list by going backwards:

```python
A = [0, 1, 2]
for elt in reversed(A):
    print(elt, end=' ')
# result: 2 1 0
```
2.5 *Itertools tools*

```python
import itertools as it
on = odd_numbers()
some_on = it.takewhile(lambda n: n < 50, on)
```

`some_on` is another generator generating all odd numbers smaller than 50.

```python
import itertools as it
on = odd_numbers()
some_on = it.islice(on, 3, 20, 3)  # start, stop index and steps
```

`list(some_on)` returns `[7, 13, 19, 25, 31, 37]`
2.6 List Filling Pattern

Common programming pattern:

```python
L = []
for k in range(n):
    L.append(some_function(k))
```

use instead:

```python
L = [some_function(k) for k in range(n)]
```

But this does not work if `function(k)` depends on earlier values of `function(j), j < k`.
2.7 Complex List Filling Pattern

```
L = [0,1]
for k in range(n):
    # call various functions here
    # that compute "result"
    L.append(result)
```

Use a generator instead:
```
def result_generator(n):
    for k in range(n):
        # call various functions here
        # that compute "result"
        yield result
```

...and if you really need a list:
```
L = list(result_generator(n)) # no append needed!
```
2.8 List from a generator

To convert a generator to a list:

```python
# for example:
g = range(10)
L = list(g)
# now L is a list with 10 elements
```
2.9 “Comprehensive” generator

Just as we had list comprehension, there are also ”comprehensive generators”:

```python
G = (n for n in range(1000) if not n % 100)
# a generator that generates 0, 100, 200, ...
```

The odd numbers again:

```python
On = (n for n in range(1000) if n%2)
```
2.10 Zipping generators

How to make one generator out of two?

```python
from itertools import izip
xg = x_generator()
yg = y_generator()
for x,y in izip(xg,yg):
    print(x,y)
```

The zipped generator stops as soon as one of the generators is exhausted.
2.11: Timings: Variants to compute the moving average of a one dimensional array

```python
class RunTimeTests(object):
    u = arange(1000000,
                dtype=float)
def ma_version1(self):
    """
    Moving average version 1
    """
    u = self.u
    self.xi = (u[:-2] +
```

Führer: FMNN25 2017
def ma_version2(self):
    
    """
    Moving average version 2
    """
    u = self.u
    self.xi = array(
        [(u[i]+u[i+1]+u[i+2])/3
        for i in range(len(u)-2)])

def ma_version3(self):
    
    """
    Moving average version 3
    """
    u = self.u
    lxi = len(u) - 2
    self.xi = empty((lxi,))
    for i in range(lxi):
        self.xi[i] = (u[i] + u[i+1] + u[i+2]) / 3.
2.12: Variants to ... - Evaluation

```
%timeit -n 5 -r 3 mktest.ma_version1()
```

5 loops, best of 3: 4.74 ms per loop

```
%timeit -n 5 -r 3 mktest.ma_version2()
```

5 loops, best of 3: 443 ms per loop

```
%timeit -n 5 -r 3 mktest.ma_version3()
```

5 loops, best of 3: 481 ms per loop

Commands with the prefix % are only available in IPython. So-called magic commands.

Führer: FMNN25 2017
2.13: Documentation

http://packages.python.org/an_example_pypi_project/sphinx.html

- Install sphinx easy_install -U sphinx or pip install sphinx

- Run sphinx-quickstart

- Make changes and run make html

Read carefully the document:
http://sphinx.pocoo.org/tutorial.html
2.14: **Tests**

In parallel to coding write tests (or even before)

Tests

- check the code
- document its used
- and document what has been tested.
2.15: Unit tests

The command

```
python -m unittest discover
```

Checks all files in a directory tree for tests and executes them.

Typical result

```
  with open(file_name, 'w') as f:
  IOError: [Errno 2] No such file or directory: '/home/claus/file.py'

Ran 8 tests in 0.042s

FAILED (errors=2)
```
2.16: Examples

```python
from bisection import bisect
import unittest

class TestIdentity(unittest.TestCase):
    def test(self):
        result = bisect(lambda x: x, -1.2, 1., tol=1.e-8)
        expected = 0.
        self.assertAlmostEqual(result, expected)

if __name__ == '__main__':
    unittest.main()
```

`bisect` is the function we want to test – the bisection method.
2.17: Examples (Cont.)

Collecting tests:

```python
import unittest

from bisection import bisect

class TestIdentity(unittest.TestCase):
    def identity_fcn(self, x):
        return x

    def test_functionality(self):
        result = bisect(self.identity_fcn, -1.2, 1., tol=1.e-8)
        expected = 0.
        self.assertAlmostEqual(result, expected)

    def test_reverse_boundaries(self):
        result = bisect(self.identity_fcn, 1., -1.)
        expected = 0.
        self.assertAlmostEqual(result, expected)

    def test_exceeded_tolerance(self):
        tol = 1.e-80
        self.assertRaises(Exception, bisect, self.identity_fcn, -1.2, 1., tol)
```

Führer: FMNN25 2017
if __name__ == '__main__':
    unittest.main()
2.18: Examples (Cont.)

The setUp method

```python
import unittest
import os  # used for, e.g., deleting files

from find_in_file import find_string, NotFoundError

class TestFindInFile(unittest.TestCase):
    def setUp(self):
        file = open('test_file.txt', 'w')
        file.write('aha ')
        file.close()
        self.file = open('test_file.txt', 'r')

    def tearDown(self):
        os.remove(self.file.name)

    def test_exists(self):
        line_no = find_string(self.file, 'aha ')
        self.assertEqual(line_no, 0)

    def test_not_exists(self):
```

Führer: FMNN25 2017
self.assertRaises(NotFoundError, find_string, self.file, 'bha')

if __name__ == '__main__':
    unittest.main()
2.19: Class Methods

An example

class Polynomial(object):
    def __init__(self, coeff):
        self.coeff = array(coeff)
    @classmethod
    def by_points(cls, x, y):
        degree = x.shape[0] - 1
        coeff = polyfit(x, y, degree)
        return cls(coeff)
    def __eq__(self, other):
        return allclose(self.coeff, other.coeff)

p1 = Polynomial.by_points(array([0., 1.]), array([0., 1.]))
p2 = Polynomial([1., 0.])

print(p1 == p2)

Note: cls refers to the class while self refers to the instance. This is caused by the decorator @classmethod.