Developing on ROS Framework

Introduction to Python

Rodrigo Ventura
João Reis
Institute for Systems and Robotics
Instituto Superior Técnico, Lisboa

Lisbon, 23-26 June 2013

Sponsored by project FP7-ICT-2011-9-601033 (MOnarCH)

The Python language

- Created in late 1980's by Guido van Rossum →

- General purpose, high-level language
- Multi-platform: UNIX, Windows, Java, embedded, etc.
- Multi-paradigm: structured, object-oriented, funcional
- Dynamic typing, automatic memory management
- Open source
- Performance:
  - on-the-fly compilation to a bytecode, which is then executed by a virtual machine
  - compiled bytecode can be cached into auxiliary files
  - just-in-time compilers to native machine code exist

The Python language

- Versions:
  - stable and still widely in use: 2.x ← *will use this one here*
  - next generation: 3.x
- Resources:
  - main website: http://www.python.org
  - documentation: http://docs.python.org/2/
    - tutorial
    - library reference ← *recommended reference documentation*
    - language reference
    - ...
- Availability
  - already built-in in Linux and Mac OS X
  - freely available installers exist for Windows

Interaction with python

- Invoking the interpreter:

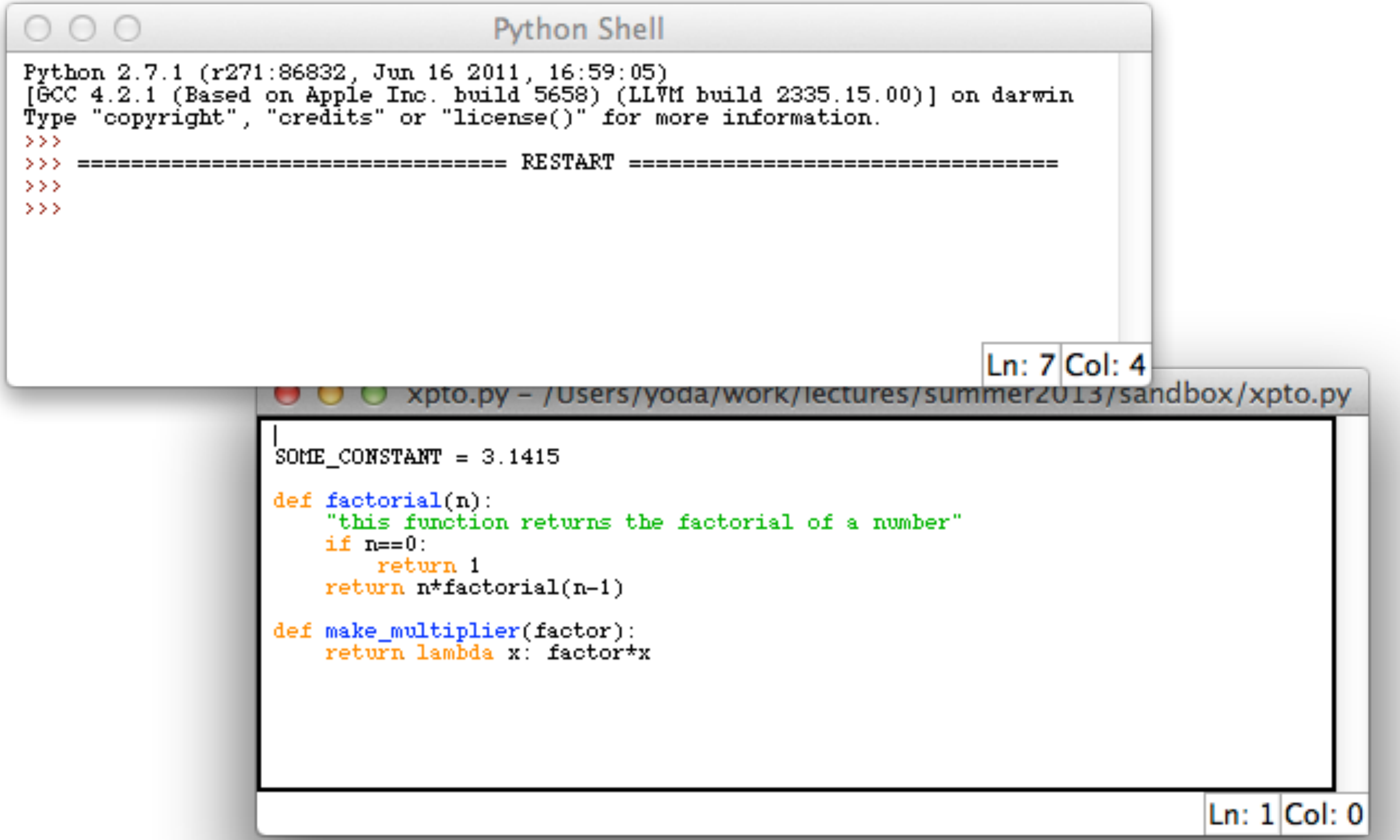
```
$ python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM
build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> _
```

  - Example interaction:

```
>>> 1+1
2
>>> print "Hello world"
Hello world
>>> _
```

Interaction with python

- IDLE: Python’s Integrated DeveLopment Environment



Python Shell
Ln: 7 Col: 4
xpto.py - /Users/yoda/work/lectures/summer2013/sandbox/xpto.py
Ln: 1 Col: 0

```
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
>>> ================================ RESTART ================================
>>>
>>>
```

```
SOME_CONSTANT = 3.1415

def factorial(n):
    "this function returns the factorial of a number"
    if n==0:
        return 1
    return n*factorial(n-1)

def make_multiplier(factor):
    return lambda x: factor*x
```

Running Python scripts

- Python files have extension **.py**
- Example:
  - file: hw.py

```
print "Hello world"
```

  - execute it with

```
$ python hw.py
Hello world
$ _
```

Executable Python scripts

- (for UNIX environments only)
  - file: hw

```
#! /usr/bin/env python
print "Hello world"
```

  - set executable flag:

```
$ chmod +x hw
```

  - execute it

```
$ ./hw
Hello world
$ _
```

Informal introduction

- Numbers: integer, float, complex

```
>>> # This is a comment
...
>>> (50-5*6)/4
5
>>> 7/3
2
>>> 7.0/3
2.3333333333333335
>>> x=20
>>> y=40
>>> x + 2*y
100
>>> 1+1j
(1+1j)
>>> (2+3j)*(4+5j)
(-7+22j)
```

Informal introduction

- Strings

```
>>> s1 = "this’s a string"
>>> s2 = 'in "quotes"'
>>> s1 + s2
'this\'s a stringin "quotes"'
>>> 2*s1
"this's a stringthis's a string"
>>> len(s1)
17
```

  - C-like formatting

```
>>> s3 = "%s is %.2f with %s significant digits"%("pi", 3.1415, 3)
>>> s3
'pi is 3.14 with 3 significant digits'
```

  - **IMPORTANT: strings are immutable**

Informal introduction

- Indexing

```
>>> s = "hello world"
>>> s[0]
'h'
>>> s[1]
'e'
>>> s[2]+s[0]
'lh'
>>> s[-1]
'd'
>>> s[-2]
'l'
```

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Informal introduction

- Slicing

```
>>> s = "hello world"
>>> s[1:5]
'ello'
>>> s[:5]
'hello'
>>> s[6:]
'world'
>>> s[-5:]
'world'
>>> s[:-6]
'hello'
```

h	e	l	l	o		w	o	r	l	d
0	1	2	3	4	5	6	7	8	9	10
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Informal introduction

- Lists

```
>>> a = ['hello', 'world', 10, 0.1]
>>> a[0]
'hello'
>>> a[-1]
0.1
>>> a[1:3]
['world', 10]
>>> len(a)
4
>>> [0, a] + 3*['x']
[0, ['hello', 'world', 10, 0.1], 'x', 'x', 'x']
>>> a[1:3] = [0]
>>> a
['hello', 0, 0.1]
```

  - **Note that lists are mutable**

Informal introduction

- Booleans and conditional expressions

```
>>> 1 == 1
True
>>> 1 != 1
False
>>> True and True
True
>>> False or True
True
>>> not True
False
>>> 'hi' == 'hi'
True
>>> 3 in [1, 2, 3]
True
>>> "Mac" in "BigMac"
True
```

Flow control

- **if** statements

```
if x < 0:
    y = 'negative'
elif x == 0:
    y = 'zero'
else:
    y = 'positive'
```

  - indentation defines blocks
  - arbitrary indentation length
  - same indentation means same block
  - a colon (:) expects a following indented block

Flow control

- **for** statements
  - runs indented block over all values of a list

```
for x in [1, 2, 3, 4, 5]:
    print 10*x
```

  - **for** statements can be nested

```
for x in [1, 2]:
    for y in [1, 2, 3]:
        for z in ['a', 'b']:
            print z
        print 100*x + y
```

Flow control

- **range** function

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(1, 11, 2)
[1, 3, 5, 7, 9]
```

```
for x in range(1, 3):
    for y in range(1, 4):
        for z in ['a', 'b']:
            print z
        print 100*x + y
```

Flow control

- **xrange** function

```
>>> wtf = xrange(10)
>>> wtf
xrange(10)
>>> type(wtf)
<type 'xrange'>
```

```
for x in xrange(1, 3):
    for y in xrange(1, 4):
        for z in ['a', 'b']:
            print z
        print 100*x + y
```

Flow control

- Miscellaneous flow control statements

```
for n in xrange(2, 10):
    for x in xrange(2, n):
        if n % x == 0:
            print n, "equals", x, '*', n/x
            break
    else:
        # no divider was found
        print n, "is prime"
```

```
while True:
    pass  # does nothing
```

```
for n in xrange(-5, 6):
    if n==0:
        continue
    print n, "is non-zero"
```

Flow control

- Defining functions
  - example:

```
def factorial(n):
    if n==0:
        return 1
    return n*factorial(n-1)

for e in xrange(3):
    n = 10**e
    f = factorial(n)
    print "factorial of %s is %s"%(n, f)
```

  - output:

```
factorial of 1 is 1
factorial of 10 is 3628800
factorial of 100 is 9332621544394415268169923885626670049071596826438162
146859296389521759999322991560894146397615651828625369792082722375825118
5210916864000000000000000000000000
```

Flow control

- defining arguments
  - multiple:

```
def draw_point(x, y):
    ...
```

  - optional:

```
def draw_point(x, y, color=”red”):
    ...

draw_point(1, 2)
draw_point(1, 2, “green”)
```

  - keyword:

```
def draw_point(x, y, color=”red”, thickness=1):
    ...

draw_point(1, 2, thickness=2)
draw_point(1, 2, thickness=2, color=”blue”)
```

Flow control

- arbitrary argument lists:

```
def printf(format, *arguments):
    ...
```

- unpacking argument lists:

```
>>> args = [3, 6]
>>> range(*args)
[3, 4, 5]
>>> range(3, 6)
[3, 4, 5]
```

Flow control

- functions as data (objects)

```
>>> range(3, 6)
[3, 4, 5]
>>> range
<built-in function range>
>>> factorial
<function factorial at 0x10d2481b8>
```

- lambda forms (a.k.a. anonymous functions):

```
>>> def make_multiplier(factor):
...     return lambda x: factor*x
...
>>> f = make_multiplier(2)
>>> f
<function <lambda> at 0x10f7ae938>
>>> print f(10)
20
```

Data structures

- Tuples are immutable lists (but more efficient)

```
>>> a = (1, 'a', True)
>>> a
(1, 'a', True)
>>> a[1]
'a'
```

- Conversion among lists, tuples, and strings

```
>>> s = "hello"
>>> list(s)
['h', 'e', 'l', 'l', 'o']
>>> tuple(list(s))
('h', 'e', 'l', 'l', 'o')
>>> str(list(s))
"['h', 'e', 'l', 'l', 'o']"
```

Data structures

- List methods

```
>>> r = range(3)
>>> r
[0, 1, 2]

>>> r.append(False)
>>> r
[0, 1, 2, False]

>>> r.extend(range(3))
>>> r
[0, 1, 2, False, 0, 1, 2]

>>> r.insert(1, True)
>>> r
[0, True, 1, 2, False, 0, 1, 2]
```

Data structures

- lists as stacks (LIFO)

```
>>> s = [1, 2]
>>> s.append(3)
>>> s.pop()
3
>>> s
[1, 2]
```

- lists as queues (FIFO)

```
>>> s = [1, 2]
>>> s.append(3)
>>> s.pop(0)
1
>>> s
[2, 3]
```

Data structures

- Functional programming

```
>>> def f(x):
...     return x%2==0
...

>>> filter(f, range(10))
[0, 2, 4, 6, 8]

>>> filter(lambda x: x%2==0, range(10))
[0, 2, 4, 6, 8]

>>> map(f, range(10))
[True, False, True, False, True, False, True, False,
True, False]

>>> map(lambda x: 2**x, range(10))
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

Data structures

- Functional programming

```
>>> def g(x, y):
...     print "g(%s, %s)"%(x, y)
...     return 2*x+y
...

>>> reduce(g, range(5))
g(0, 1)
g(1, 2)
g(4, 3)
g(11, 4)
26

>>> reduce(lambda x,y: x+y, range(1, 11))
55
>>> sum(range(1,11))
55
```

Data structures

- List comprehension

```
>>> [2**n for n in range(10)]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

>>> [2**n for n in range(10) if n%2!=0]
[2, 8, 32, 128, 512]

>>> [(n, 2**n) for n in range(10)]
[(0, 1), (1, 2), (2, 4), (3, 8), (4, 16), (5, 32), (6,
64), (7, 128), (8, 256), (9, 512)]

>>> [(x, y) for x in range(2) for y in range(3)]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

Data structures

- **del** statement

```
>>> a = list("hello world")
>>> a
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']

>>> del a[4:7]  # this is the same as a[4:7]=[]
>>> a
['h', 'e', 'l', 'l', 'o', 'r', 'l', 'd']

>>> del a[:]  # this slice includes the whole list
>>> a
[]

>>> del a
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Data structures

- Sequences are:
  - lists
  - tuples
  - strings
- tuple packing and unpacking

```
>>> t = "this", "is", "cool"
>>> t
('this', 'is', 'cool')

>>> x, y = [1, 2]
>>> x, y = y, x
>>> "x=%s y=%s"%(x, y)
'x=2 y=1 z=3'
>>> def point((x, y), color=red):
...
```

Data structures

- Operations on immutable sequences

Operation	**Result**
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n, n * s`	*n* shallow copies of *s* concatenated
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(i)`	index of the first occurrence of *i* in *s*
`s.count(i)`	total number of occurrences of *i* in *s*

Data structures

- Additional operations on mutable sequences

Operation	**Result**
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	same as `s[len(s):len(s)] = [x]`
`s.extend(x)`	same as `s[len(s):len(s)] = x`
`s.count(x)`	return number of *i*'s for which `s[i] == x`
`s.index(x[, i[, j]])`	return smallest *k* such that `s[k] == x` and `i <= k < j`
`s.insert(i, x)`	same as `s[i:i] = [x]`
`s.pop([i])`	same as `x = s[i]; del s[i]; return x`
`s.remove(x)`	same as `del s[s.index(x)]`
`s.reverse()`	reverses the items of *s* in place
`s.sort([cmp[, key[, reverse]]])`	sort the items of *s* in place

Data structures

- Sets

```
>>> a = set([1, 2, 3])
>>> a
set([1, 2, 3])

>>> b = set("hello")
>>> b
set(['h', 'e', 'l', 'o'])

>>> b - set("aeiou")
set(['h', 'l'])

>>> b & set("world")
set(['l', 'o'])

>>> b | set("world")
set(['e', 'd', 'h', 'l', 'o', 'r', 'w'])
```

Data structures

- Dictionaries (a.k.a. hash table)

```
>>> tel = { "yoda": 2195, "pal": 2274, "jseq": 2057 }
>>> tel["yoda"]
2195

>>> tel["yoda"] = "none"
>>> tel
{'jseq': 2057, 'pal': 2274, 'yoda': 'none'}

>>> tel["jjss"] = 2288
>>> tel
{'jjss': 2288, 'jseq': 2057, 'pal': 2274, 'yoda': 'none'}

>>> tel.keys()
['jjss', 'jseq', 'pal', 'yoda']

>>> tel.has_key("jseq")
True

>>> "pal" in tel
True
```

Data structures

- NOTE: dictionary maps immutable keys to arbitrary objects

```
>>> points = { (1,2): "robot", (2,3): ["box", (0,0)], "wtf":
dict(w="what", t="the") }
>>> points
{(1, 2): 'robot', (2, 3): ['box', (0, 0)], 'wtf': {'t': 'the', 'w':
'what'}}

>>> points.keys()
[(1, 2), (2, 3), 'wtf']

>>> points[(2,3)]
['box', (0, 0)]

>>> points[(2,3)][1]
(0, 0)

>>> points[(2,3)][1][0]
0

>>> points["wtf"]["f"]="*"
>>> points
{(1, 2): 'robot', (2, 3): ['box', (0, 0)], 'wtf': {'t': 'the', 'w':
'what', 'f': '*'}}
```

Data structures

- alternative ways of constructing dictionaries

```
>>> dict(yoda=2195, pal=2274, jseq=2057)
{'jseq': 2057, 'pal': 2274, 'yoda': 2195}

>>> dict( [ ("yoda", 2195), ["pal", 2274], ("jseq", 2057) ] )
{'jseq': 2057, 'pal': 2274, 'yoda': 2195}

>>> dict( [ (n,2**n) for n in xrange(10) ] )
{0: 1, 1: 2, 2: 4, 3: 8, 4: 16, 5: 32, 6: 64, 7: 128, 8: 256, 9: 512}

>>> dict( [ (b, dict( [ (n, b**n) for n in xrange(10) ] )) for b in
range(1, 4) ] )
{1: {0: 1, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1},
 2: {0: 1, 1: 2, 2: 4, 3: 8, 4: 16, 5: 32, 6: 64, 7: 128, 8: 256, 9:
512},
 3: {0: 1, 1: 3, 2: 9, 3: 27, 4: 81, 5: 243, 6: 729, 7: 2187, 8: 6561,
9: 19683}}
```

Data structures

- Looping techniques

```
>>> tel = dict(yoda=2195, pal=2274, jseq=2057)
>>> tel.items()
[('jseq', 2057), ('pal', 2274), ('yoda', 2195)]
>>> for (k,v) in tel.items():  # .iteritems() is more memory efficient
...     print k, "=", v
...
jseq = 2057
pal = 2274
yoda = 2195

>>> for (i,v) in enumerate(["a", "b", "c"]):
...     print i, v
...
0 a
1 b
2 c

>>> for (a,b) in zip(["a", "b", "c"], ["A", "B", "C"]):
...     print a, b
...
a A
b B
c C
```

Data structures

- The two faces of equality

```
>>> a = dict(a="alpha", b="bravo", c="charlie")
>>> b = dict(a="alpha", b="bravo", c="charlie")
>>> c = b

>>> a == b
True

>>> b == c
True

>>> a is b
False

>>> b is c
True
```

Modules

- A module is a collection of definitions
- Any .py source file is a module
  - example: file xpto.py

```
SOME_CONSTANT = 3.1415

def factorial(n):
    if n==0:
        return 1
    return n*factorial(n-1)
```

importing and using the xpto module

```
>>> import xpto
>>> xpto.factorial(5)
120
>>> xpto.SOME_CONSTANT
3.1415
```

Modules

- more on modules:
  - introspection:

```
>>> import xpto
>>> xpto
<module 'xpto' from 'xpto.pyc'>
>>> dir(xpto)
['SOME_CONSTANT', '__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'factorial', 'make_multiplier']
>>> xpto.__file__
'xpto.pyc'
>>> xpto.__name__
'xpto'
```

  - selective import:

```
>>> from xpto import SOME_CONSTANT
>>> SOME_CONSTANT
3.1415

>>> from xpto import *
>>> factorial(5)
120
```

Modules

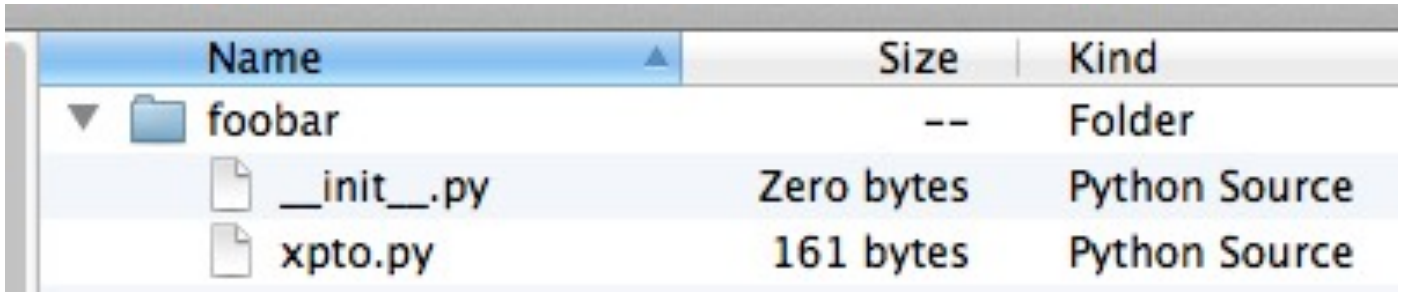
- module reload

```
>>> import xpto
>>> xpto.SOME_CONSTANT
3.1415
# At this point the xpto.py file is edited and SOME_CONSTANT is modified
>>> reload(xpto)
<module 'xpto' from 'xpto.py'>
>>> xpto.SOME_CONSTANT
999
```

```
>>> import xpto
>>> from xpto import *
>>> SOME_CONSTANT
999
# xpto.py edited and SOME_CONSTANT modified to former value
>>> reload(xpto)
<module 'xpto' from 'xpto.py'>
>>> from xpto import *
>>> SOME_CONSTANT
3.1415
```

Modules

- more on modules:
  - sub-modules:

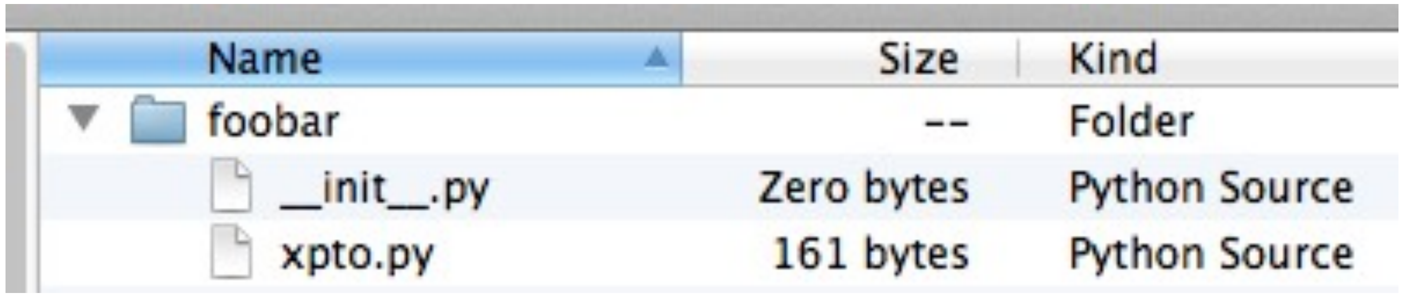


Name	Size	Kind
foobar	--	Folder
__init__.py	Zero bytes	Python Source
xpto.py	161 bytes	Python Source

```
>>> import foobar.xpto
>>> dir(foobar.xpto)
['SOME_CONSTANT', '__builtins__', '__doc__', '__file__', '__name__',
'__package__', 'factorial', 'make_multiplier']

>>> foobar.xpto.SOME_CONSTANT
3.1415

>>> import foobar.xpto as x
>>> x.SOME_CONSTANT
3.1415
```

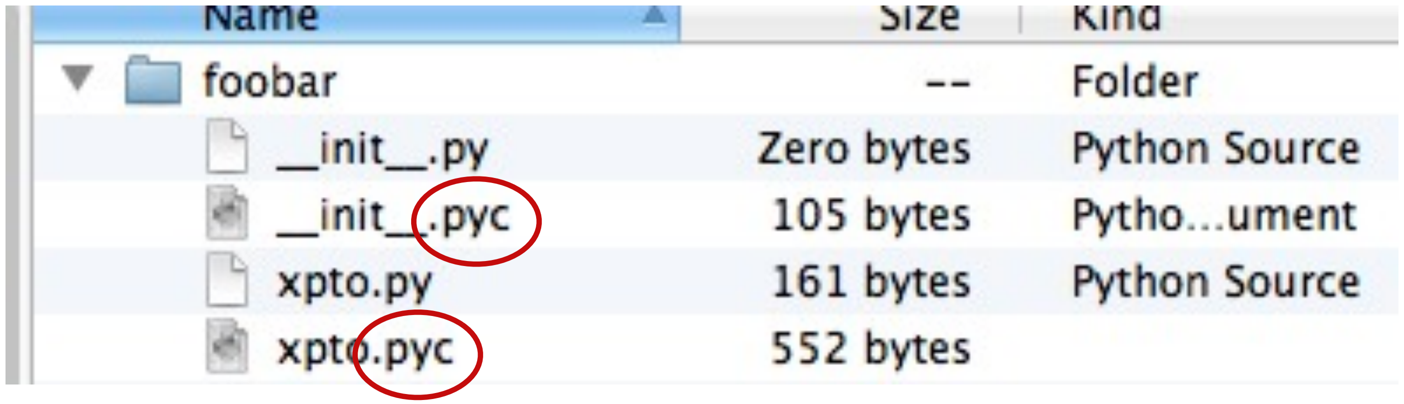
Modules

- module compilation into bytecode upon import



Name	Size	Kind
foobar	--	Folder
__init__.py	Zero bytes	Python Source
xpto.py	161 bytes	Python Source

```
>>> import foobar.xpto
```



Name	Size	Kind
foobar	--	Folder
__init__.py	Zero bytes	Python Source
__init__.pyc	105 bytes	Pytho...ument
xpto.py	161 bytes	Python Source
xpto.pyc	552 bytes	

Inline documentation

- if first line of a function is a string, it becomes its documentation

```
def factorial(n):
    "this function returns the factorial of a number"
    if n==0:
        return 1
    return n*factorial(n-1)
```

```
>>> import xpto
>>> xpto.factorial
<function factorial at 0x1004bc9b0>
>>> help(xpto.factorial)
Help on function factorial in module xpto:

factorial(n)
    this function returns the factorial of a number
```

Input and Output

- Basic file operations
  - reading

```
>>> fh = open("xpto.py")
>>> fh
<open file 'xpto.py', mode 'r' at 0x1004a9d20>

>>> fh.read(5)
'\nSOME'

>>> fh.readline()
'_CONSTANT = 3.1415\n'

>>> fh.readlines()
['\n', 'def factorial(n):\n', '    if n==0:\n', '        return 1\n', '
return n*factorial(n-1)\n', '\n', 'def make_multiplier(factor):\n', '
return lambda x: factor*x\n']

>>> [len(ln) for ln in open("xpto.py")]
[1, 23, 1, 18, 13, 17, 28, 1, 29, 30]
```

Input and Output

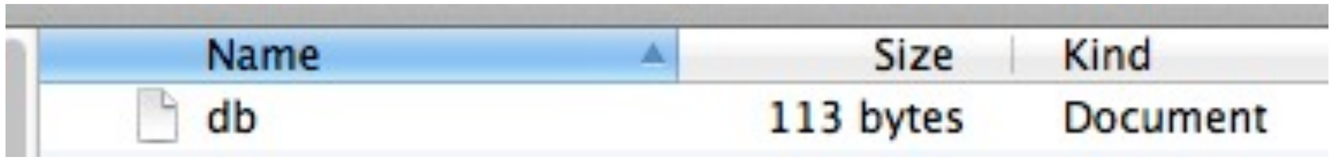
- Basic file operations
  - writing

```
>>> fh = open("trash", "w")
>>> fh
<open file 'trash', mode 'w' at 0x1004a9db0>

>>> fh.write("xpto")
>>> print >>fh, 1+1
>>> print >>fh, "end of file"
>>> fh.close()

>>> print open("trash").read()
xpto2
end of file

>>> with open("trash") as fh:
...     print [len(ln) for ln in fh]
...
[6, 12]
```

Input and Output

- Serialization: transformation between data structures and portable sequences of bytes

```
>>> import pickle

>>> data = dict(a="alpha", b="bravo", c="charlie", d="delta", e="echo")
>>> data
{'a': 'alpha', 'c': 'charlie', 'b': 'bravo', 'e': 'echo', 'd': 'delta'}

>>> with open("db", "w") as fh:
...     pickle.dump(data, fh)
```



Name	Size	Kind
db	113 bytes	Document

```
>>> import pickle
>>> with open("db") as fh:
...     recover = pickle.load(fh)
...
>>> recover
{'a': 'alpha', 'c': 'charlie', 'b': 'bravo', 'e': 'echo', 'd': 'delta'}
```

Exceptions

- Exceptions are errors detected at run time

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero

>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined

>>> "2" + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

Exceptions

- Exceptions can be gracefully handled in run time

```
>>> try:
...       print 10 * (1/0)
... except ZeroDivisionError:
...       print "Bummer!"
...
Bummer!
```

and raised if necessary

```
>>> try:
...       print 10 * (1/2)
...       raise ZeroDivisionError
... except ZeroDivisionError:
...       print "Fooled you!"
...
0
Fooled you!
```

Exceptions

- A more sophisticated example

```
fh = None
try:
    fh = open("somefile")
    ln = fh.readline()
    x  = int(ln.strip())
except IOError:
    print "Can't open file"
except ValueError:
    print "Can't extact integer"
except:
    print "Unknown error"
else:
    print "All went well"
finally:
    if fh is not None:
        fh.close()
```

Classes and Objects

- A class in python encapsulates a set of statements

```
# filename: ypto.py
class MyClass:
    s = "hello world"

    def getter(self):
        return self.s

    def setter(self, value):
        self.s = value
```

```
>>> from ypto import *
>>> obj = MyClass()
>>> obj
<ypto.MyClass instance at 0x100541a28>

>>> obj.getter()
'hello world'

>>> obj.setter("foo bar")
>>> obj.getter()
'foo bar'
```

Classes and Objects

- Defining constructors: __init__() method

```
class MyClass2:

    def __init__(self):
        self.s = "hello world"

    ...
```

  - passing arguments to the constructor:

```
class MyClass3:

    def __init__(self, initial):
        self.s = initial

    ...
```

```
>>> obj = MyClass3("a")
>>> obj.getter()
'a'
```

Classes and Objects

- Bound and unbound methods

```
>>> MyClass3
<class ypto.MyClass3 at 0x1086518d8>

>>> obj = MyClass3("a")
>>> obj
<ypto.MyClass3 instance at 0x1086729e0>

>>> MyClass3.getter
<unbound method MyClass3.getter>

>>> obj.getter
<bound method MyClass3.getter of <ypto.MyClass3 instance at
0x1086729e0>>

>>> obj.getter()
'a'

>>> MyClass3.getter(obj)
'a'

>>> f = obj.getter
>>> f()
'a'
```

Classes and Objects

- Inheritance

```
# deriving MyClass4 from MyClass
class MyClass4(MyClass):

    def __init__(self, initial):
        self.s = initial

    def getter(self):
        print "getting"
        return self.s
```

```
>>> obj = MyClass4(0)

>>> obj.setter(3)

>>> obj.getter()
getting
3
```

Classes and Objects

- More on inheritance

```
class MyClass5(MyClass2):
    def __init__(self, initial=None):
        if initial is None:
            MyClass2.__init__(self)
        else:
            self.s = initial
```

```
>>> o1 = MyClass5()
>>> o1.getter()
'hello world'

>>> o2 = MyClass5(2)
>>> o2.getter()
2
```

Classes and Objects

- Multiple inheritance

```
class A:
    def a(self):
        print "I'm A"

class B:
    def b(self):
        print "I'm B"

class C(A, B):
    def c(self):
        self.a()
        self.b()
        print "And I'm C"
```

```
>>> obj = C()
>>> obj.c()
I'm A
I'm B
And I'm C
```

Classes and Objects

- Iterator is an object capable of iterating over a range of values, not necessarily explicit in memory

```
>>> a = [1, "a", (0,0), False]
>>> i = iter(a)
>>> i
<listiterator object at 0x10d227190>
>>> i.next()
1
>>> i.next()
'a'
>>> i.next()
(0, 0)
>>> i.next()
False
>>> i.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Classes and Objects

- sequences, xrange’s, dictionaries, sets, files, all share an iterator interface, i.e., iter() yields
- the **for** statement only requires an object with iterator interface; that’s why **for** works with all of the above types without knowing about their internals
- you can add iterator interfaces to your own classes

```
class MyXRange:
    def __init__(self, count):
        self.count = count

    def __iter__(self):
        self.n = -1
        return self

    def next(self):
        if self.n<self.count-1:
            self.n += 1
            return self.n
        else:
            raise StopIteration
```

Classes and Objects

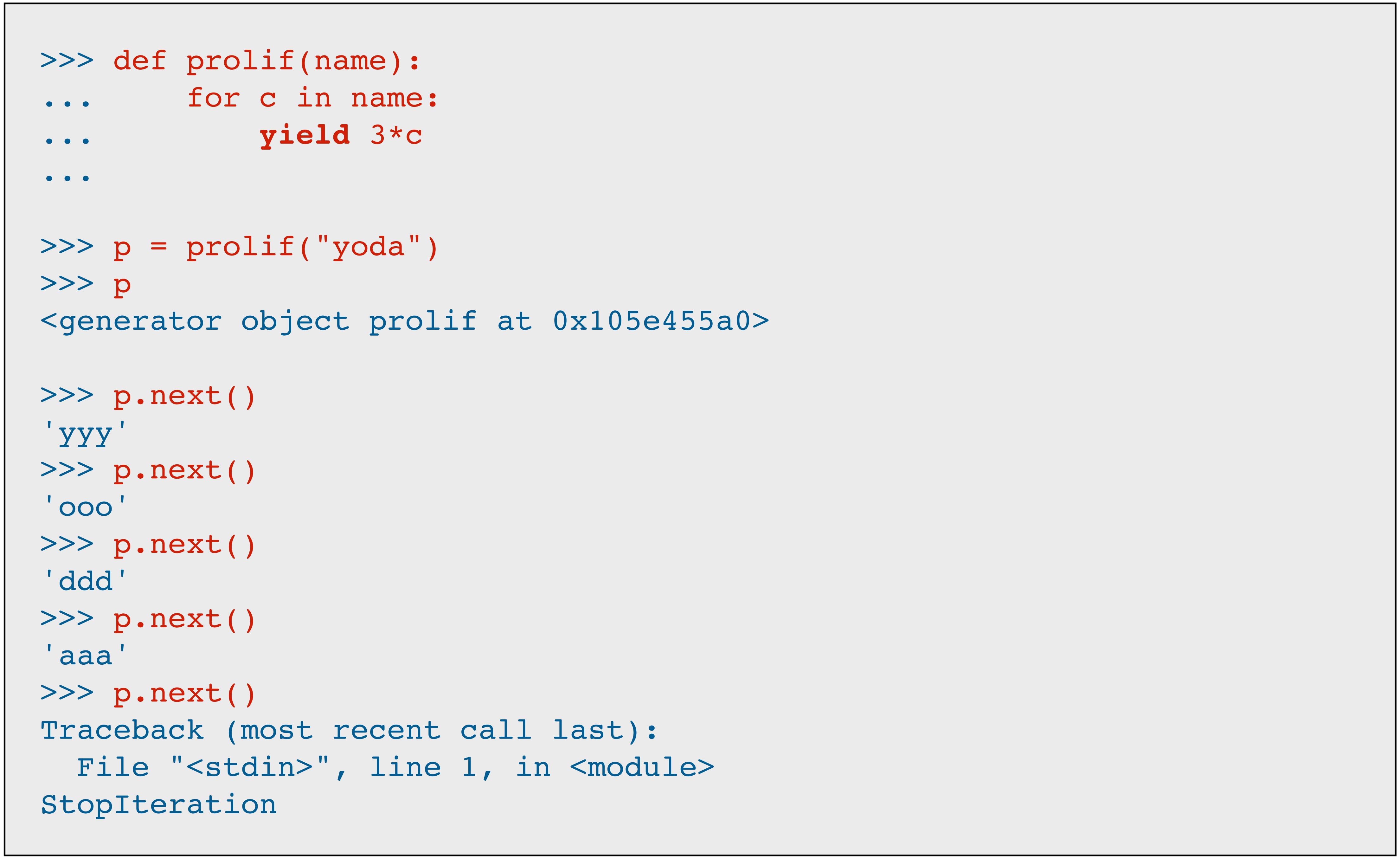
- note the difference:

```
>>> [ 2**n for n in xrange(10) ]
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

>>> ( 2**n for n in xrange(10) )
<generator object <genexpr> at 0x105e455a0>

>>> g = ( 2**n for n in xrange(10) )
>>> i = iter(g)
>>> i.next()
1
>>> i.next()
2
...
```

the second form does not create the list in memory, but rather computes values on demand via the iterator

Classes and Objects

- Generators are powerful tools to create iterators



```
>>> def prolif(name):
...     for c in name:
...         yield 3*c
...

>>> p = prolif("yoda")
>>> p
<generator object prolif at 0x105e455a0>

>>> p.next()
'yyy'
>>> p.next()
'ooo'
>>> p.next()
'ddd'
>>> p.next()
'aaa'
>>> p.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Classes and Objects

- Iterators can be used in many other situations

```
>>> prolif("yoda")
<generator object prolif at 0x105e455a0>

>>> [x for x in prolif("yoda")]
['yyy', 'ooo', 'ddd', 'aaa']

>>> list(prolif("yoda"))
['yyy', 'ooo', 'ddd', 'aaa']

>>> tuple(prolif("yoda"))
('yyy', 'ooo', 'ddd', 'aaa')

>>> set(prolif("yoda"))
set(['aaa', 'ooo', 'yyy', 'ddd'])
```

Examples from standard library

- Command line arguments

```
# filename: aaa.py
import sys
print sys.argv
```

```
$ python aaa.py
['aaa.py']

$ python aaa.py first second third
['aaa.py', 'first', 'second', 'third']
```

Examples from standard library

- Operating system interface

```
>>> import os

>>> os.getcwd()
'/Users/yoda/work/lectures/summer2013/sandbox'

>>> os.chdir('/usr/local')

>>> os.system('ls')
CONTRIBUTING.md            _EVIL_LIBS
info                       opencv
[...]
SUPPORTERS.md              include
ocaml
0
```

Examples from standard library

- Mathematics

```
>>> import math
>>> math.cos(math.pi / 4.0)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(xrange(100), 10)
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()
0.17970987693706186
>>> random.randrange(6)
4
```

Examples from standard library

- Internet access

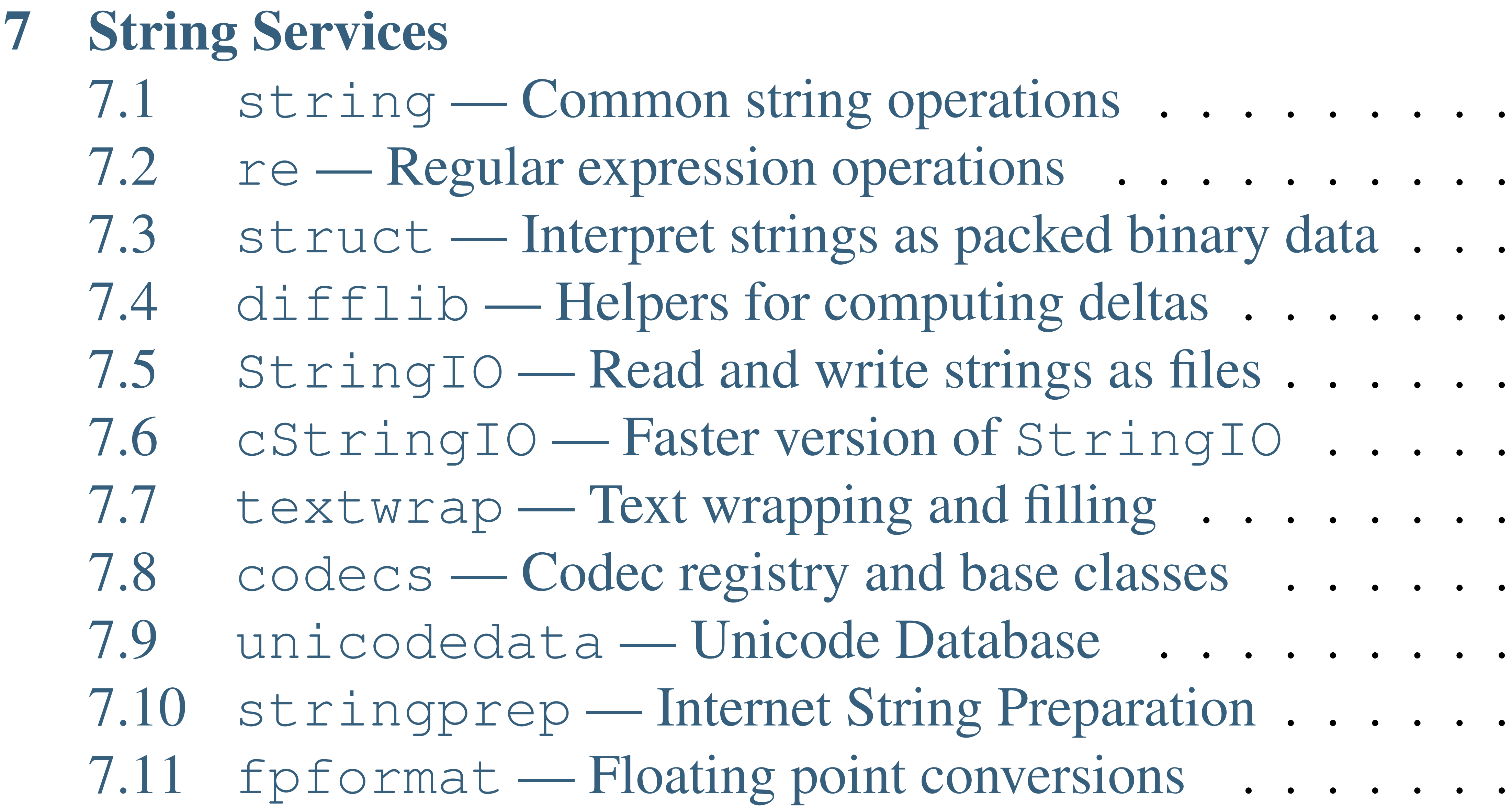
```
>>> import urllib2

>>> fh = urllib2.urlopen('http://www.ist.utl.pt')

>>> fh.readline()
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//
EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">\n'
>>> fh.readline()
'<html xmlns="http://www.w3.org/1999/xhtml"
xml:lang="pt" lang="pt">\n'
>>> fh.readline()
'<head>\n'
>>> fh.readline()
'<title>T\xc3\xa9cnico Lisboa - Engenharia,
Arquitectura, Ci\xc3\xaancia e Tecnologia</title>\n'
```

Examples from standard library

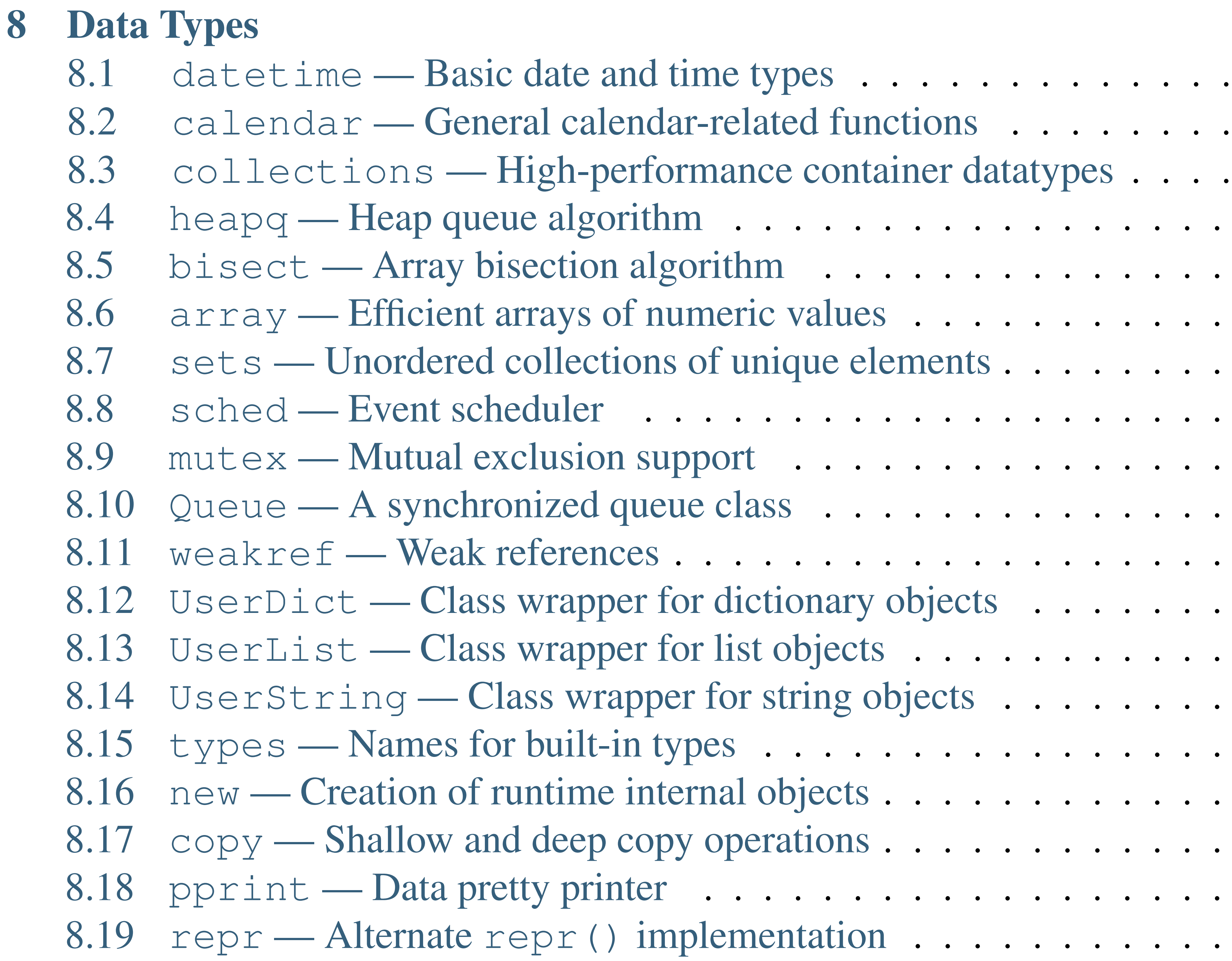
- Date and time

```
>>> import datetime

>>> datetime.datetime.now()
datetime.datetime(2013, 7, 23, 17, 29, 58, 42557)

>>> today = datetime.date.today()
>>> birthday = datetime.date(1973, 10, 31)
>>> age = today - birthday

>>> age
datetime.timedelta(14510)

>>> age.days
14510
```

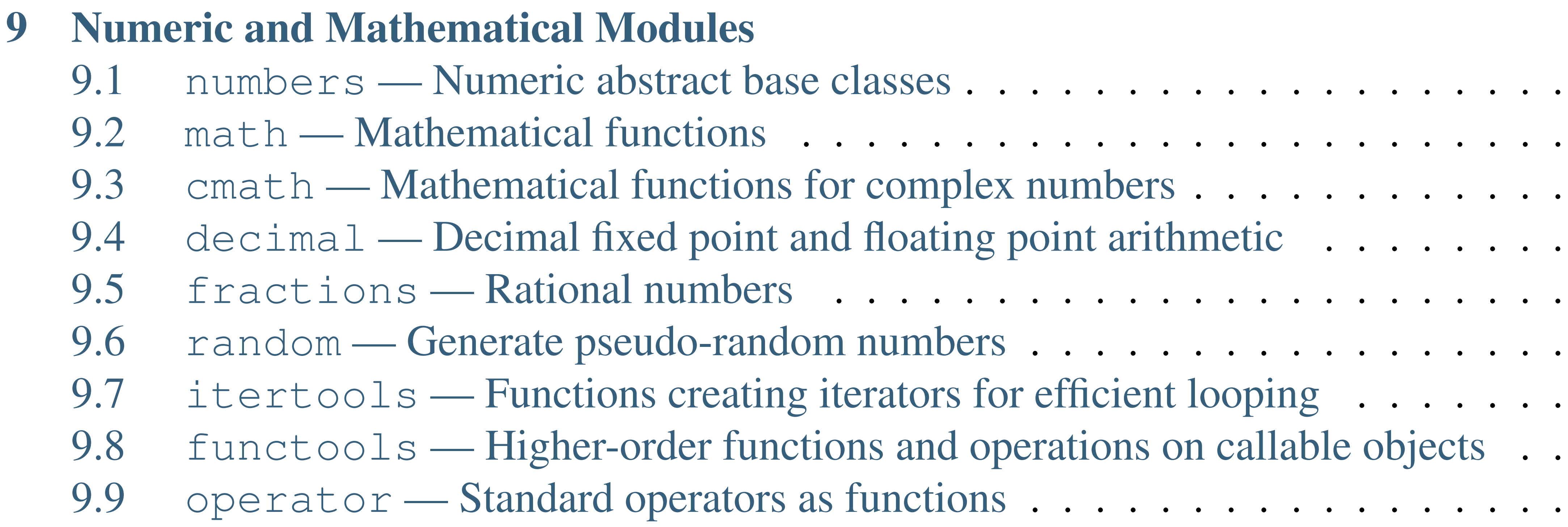
Examples from standard library



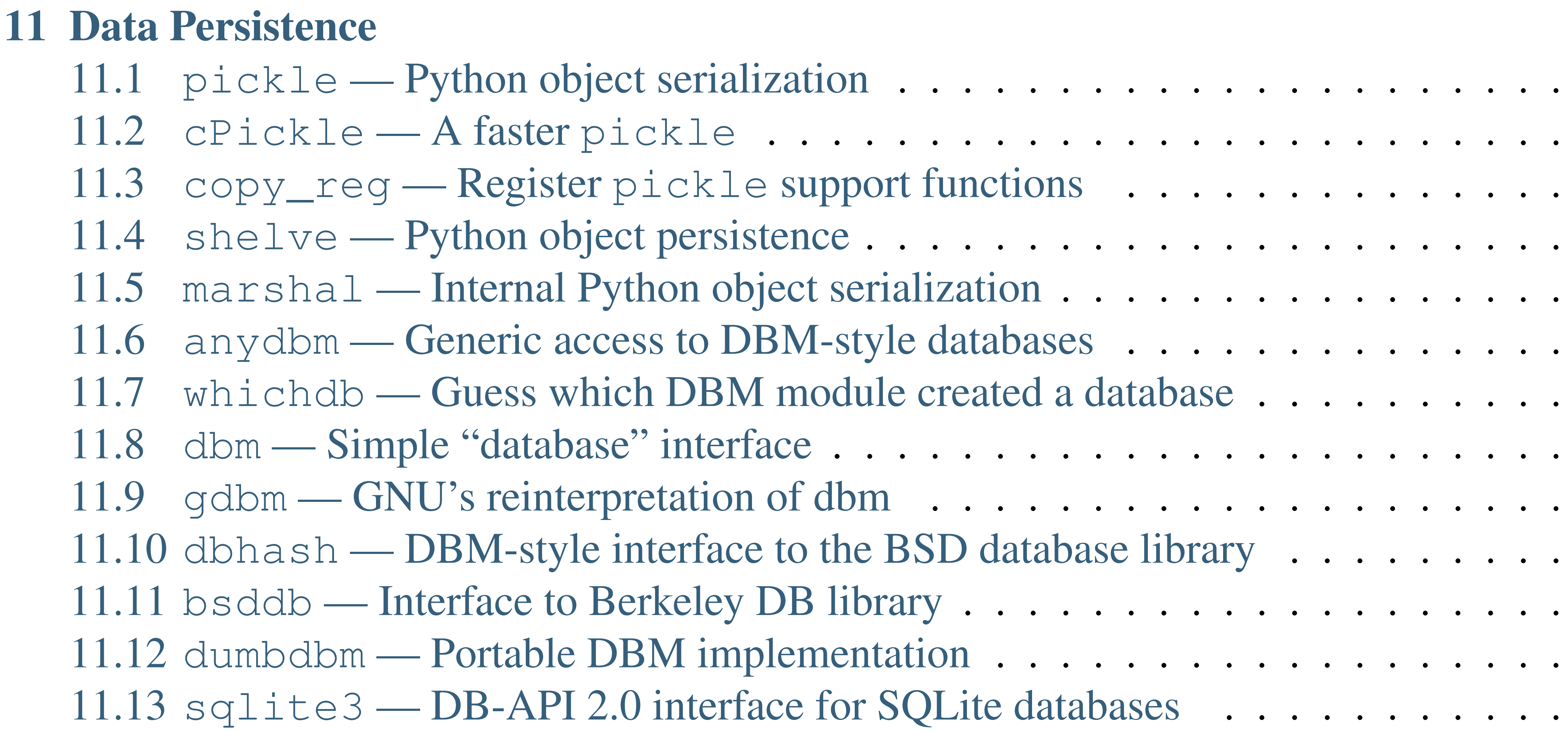
Examples from standard library



Examples from standard library

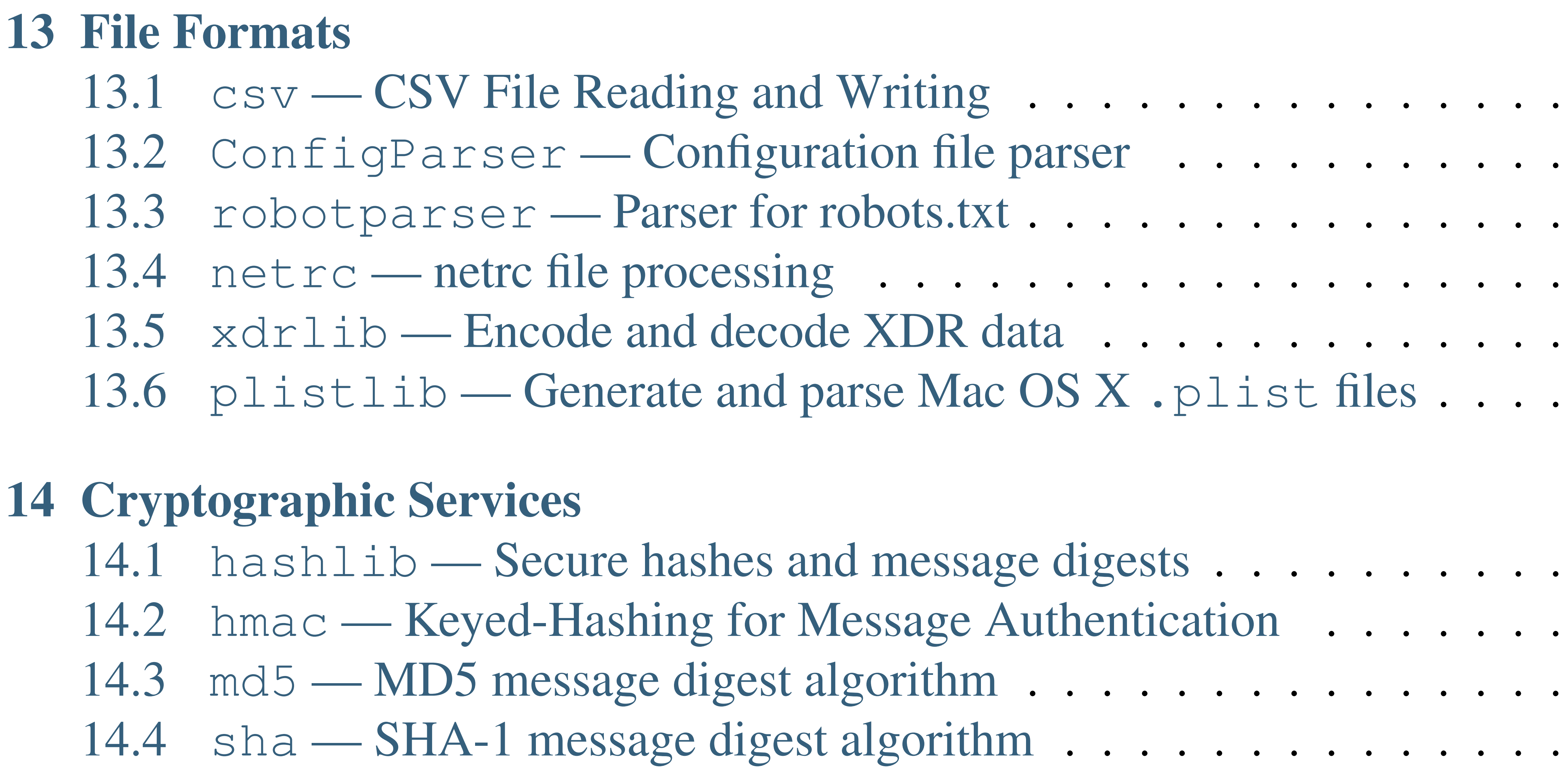


Examples from standard library

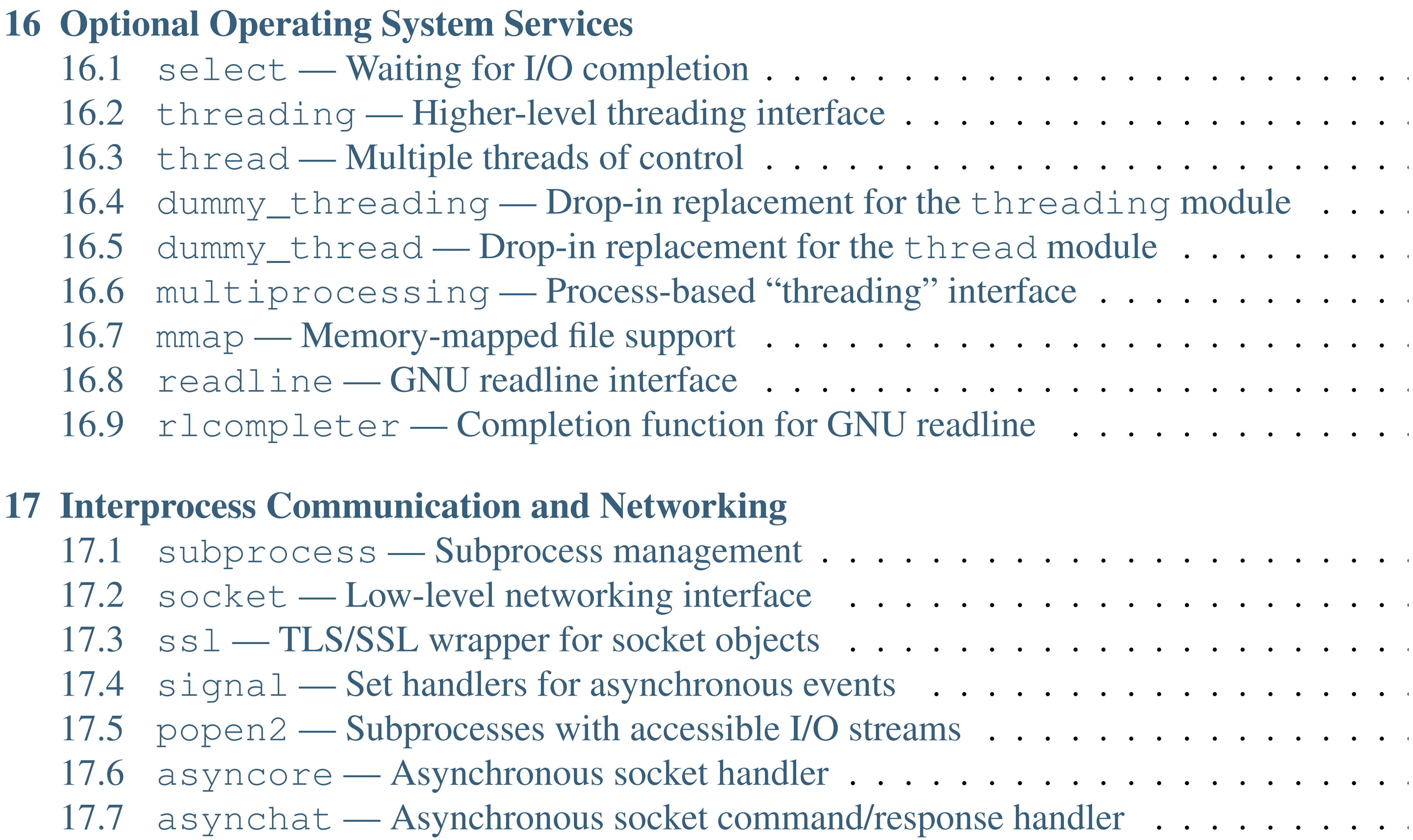


11 Data Persistence

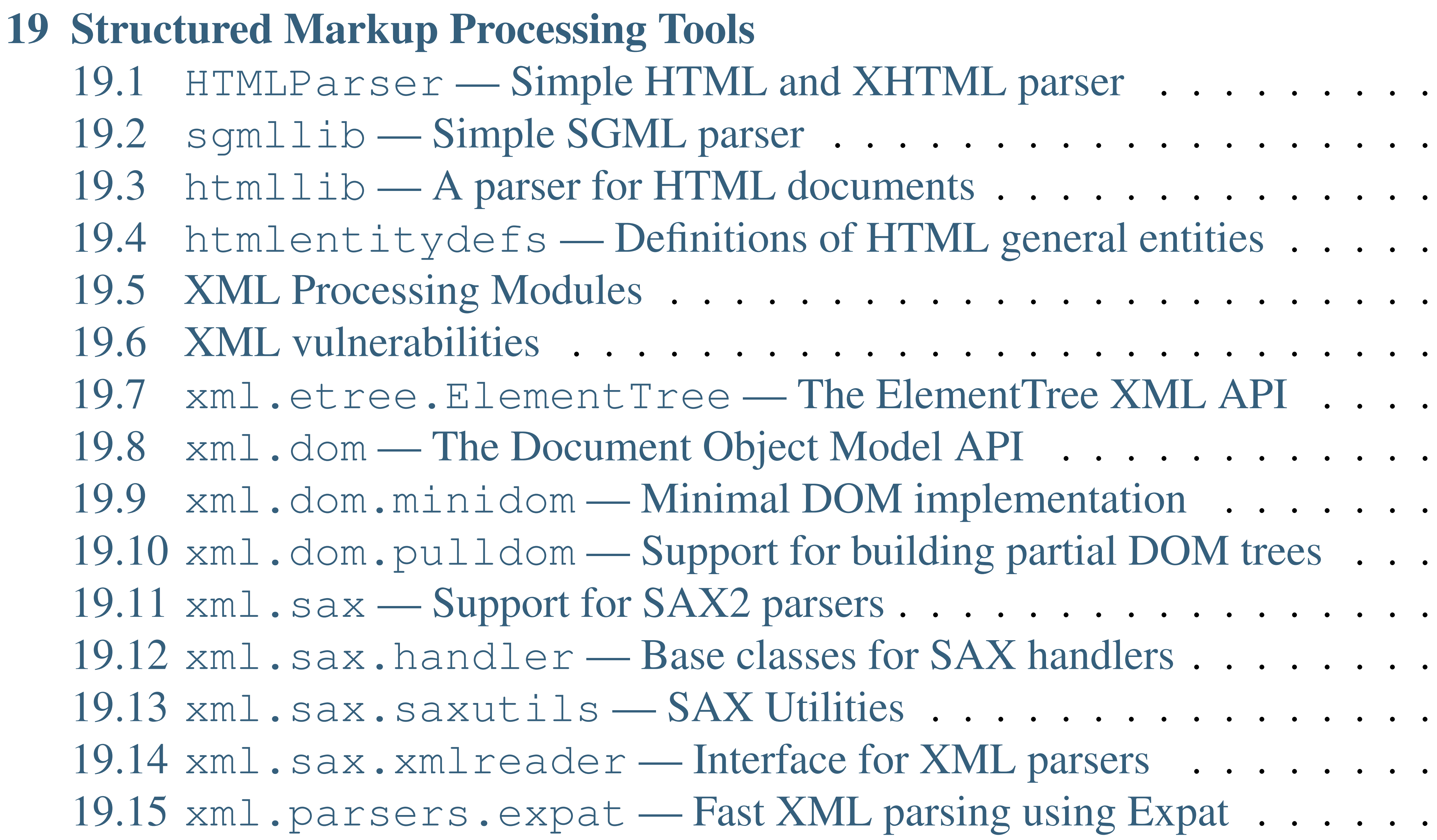
Examples from standard library



Examples from standard library



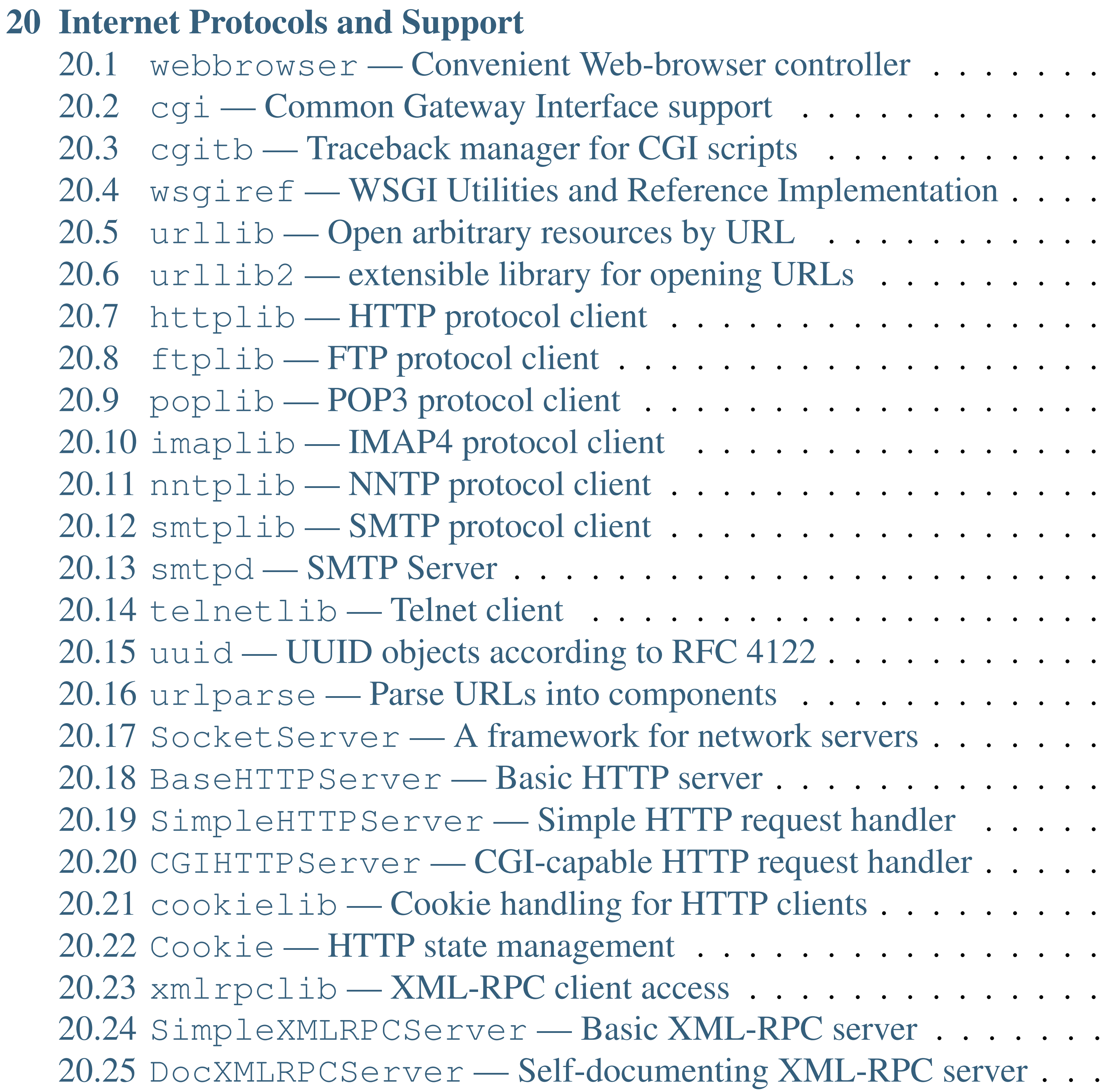
Examples from standard library



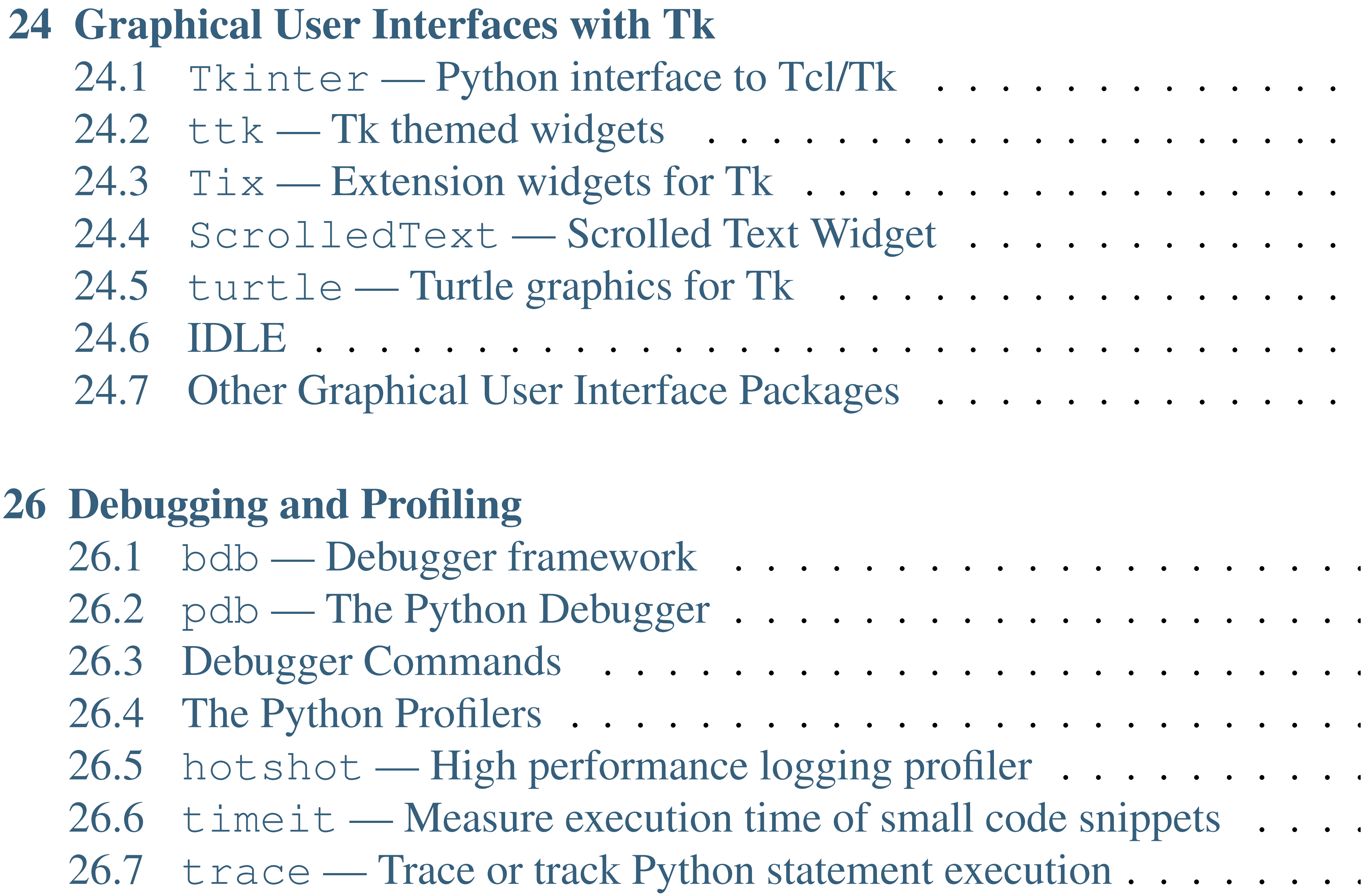
19 Structured Markup Processing Tools

19.1 `HTMLParser` — Simple HTML and XHTML parser
19.2 `sgmllib` — Simple SGML parser
19.3 `htmllib` — A parser for HTML documents
19.4 `htmlentitydefs` — Definitions of HTML general entities
19.5 XML Processing Modules
19.6 XML vulnerabilities
19.7 `xml.etree.ElementTree` — The ElementTree XML API
19.8 `xml.dom` — The Document Object Model API
19.9 `xml.dom.minidom` — Minimal DOM implementation
19.10 `xml.dom.pulldom` — Support for building partial DOM trees
19.11 `xml.sax` — Support for SAX2 parsers
19.12 `xml.sax.handler` — Base classes for SAX handlers
19.13 `xml.sax.saxutils` — SAX Utilities
19.14 `xml.sax.xmlreader` — Interface for XML parsers
19.15 `xml.parsers.expat` — Fast XML parsing using Expat

Examples from standard library



Examples from standard library



Packages central repository

- Thousands of third-party packages can be found on the main package repository here:

  http://pypi.python.org

- Command line tool to manage packages: **pip**

```
Usage:
  pip <command> [options]

Commands:
  install                     Install packages.
  uninstall                   Uninstall packages.
  freeze                      Output installed packages in requirements format.
  list                        List installed packages.
  show                        Show information about installed packages.
  search                      Search PyPI for packages.
  zip                         Zip individual packages.
  unzip                       Unzip individual packages.
  bundle                      Create pybundles.
  help                        Show help for commands.
```

SciPy project

SciPy.org Sponsored By ENTHOUGHT

- Numerical computing framework for Python
- Install with **pip**

- SciPy comprises several sub-projects:
  - NumPy: n-dimensional arrays and basic matrix routines
  - SciPy: library for scientific computing
  - Matplotlib: 2D and 3D plotting routines
  - IPython: powerful interactive console
  - Sympy: symbolic mathematics
  - pandas: data analysis

Introduction to NumPy

- Creating arrays

```
>>> from numpy import *
>>> array([1, 2, 3, 4])
array([1, 2, 3, 4])
>>> array([[1,2], [3,4]])
array([[1, 2],
       [3, 4]])
>>> array([[[1,2], [3,4]], [[5,6], [7,8]]])
array([[[1, 2],
        [3, 4]],

       [[5, 6],
        [7, 8]]])
>>> eye(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> zeros(2)
array([ 0.,  0.])
>>> zeros((2,2))
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> ones((2,3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Introduction to NumPy

- Indexing arrays

```
>>> a = array([[1,2,3], [4,5,6], [7,8,9]])
>>> a
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

>>> a[0,2]
3

>>> a[1]
array([4, 5, 6])

>>> a[:,1]
array([2, 5, 8])

>>> a[1:3,1:2]
array([[5],
       [8]])
```

Introduction to NumPy

- Working with axes

```
>>> a.shape
(3, 3)

>>> a.T
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])

>>> b = array([1,2,3,4])

>>> b[None,:]
array([[1, 2, 3, 4]])
>>> b[None,:].shape
(1, 4)

>>> b[:,None]
array([[1],
       [2],
       [3],
       [4]])
>>> b[:,None].shape
(4, 1)
```

Introduction to NumPy

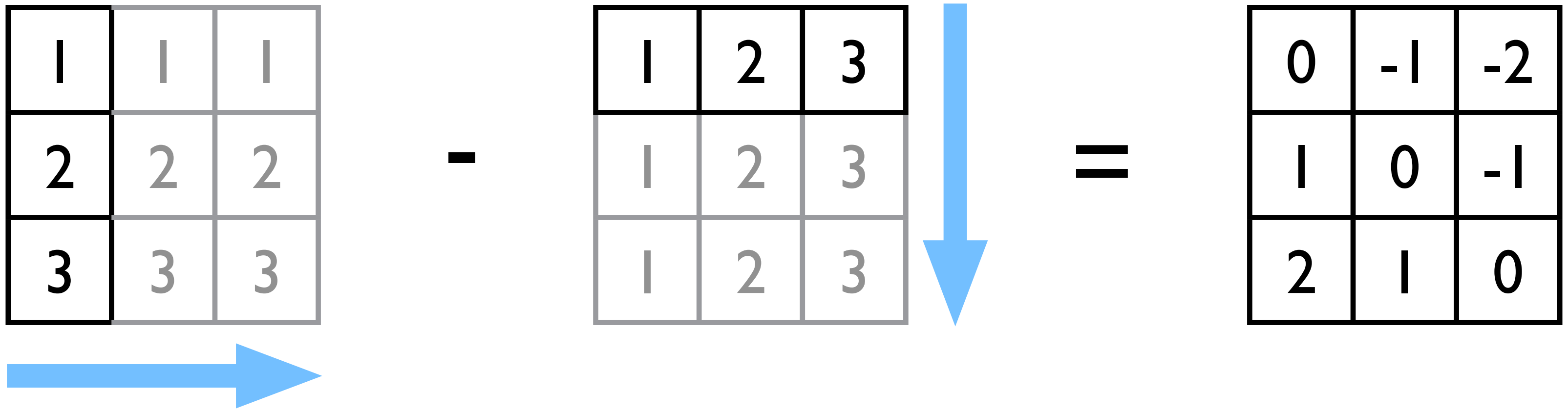
- Operations with arrays

```
>>> a + eye(3)
array([[  2.,   2.,   3.],
       [  4.,   6.,   6.],
       [  7.,   8.,  10.]])

>>> a + 2*eye(3)
array([[  3.,   2.,   3.],
       [  4.,   7.,   6.],
       [  7.,   8.,  11.]])

>>> eye(3)*a
array([[ 1.,  0.,  0.],
       [ 0.,  5.,  0.],
       [ 0.,  0.,  9.]])

>>> dot(eye(3), a)
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])
```

Introduction to NumPy

- Broadcasting

```
>>> c = array([1,2,3])

>>> c[:,None]
array([[1],
       [2],
       [3]])

>>> c[None,:]
array([[1, 2, 3]])

>>> c[:,None] - c[None,:]
array([[ 0, -1, -2],
       [ 1,  0, -1],
       [ 2,  1,  0]])
```



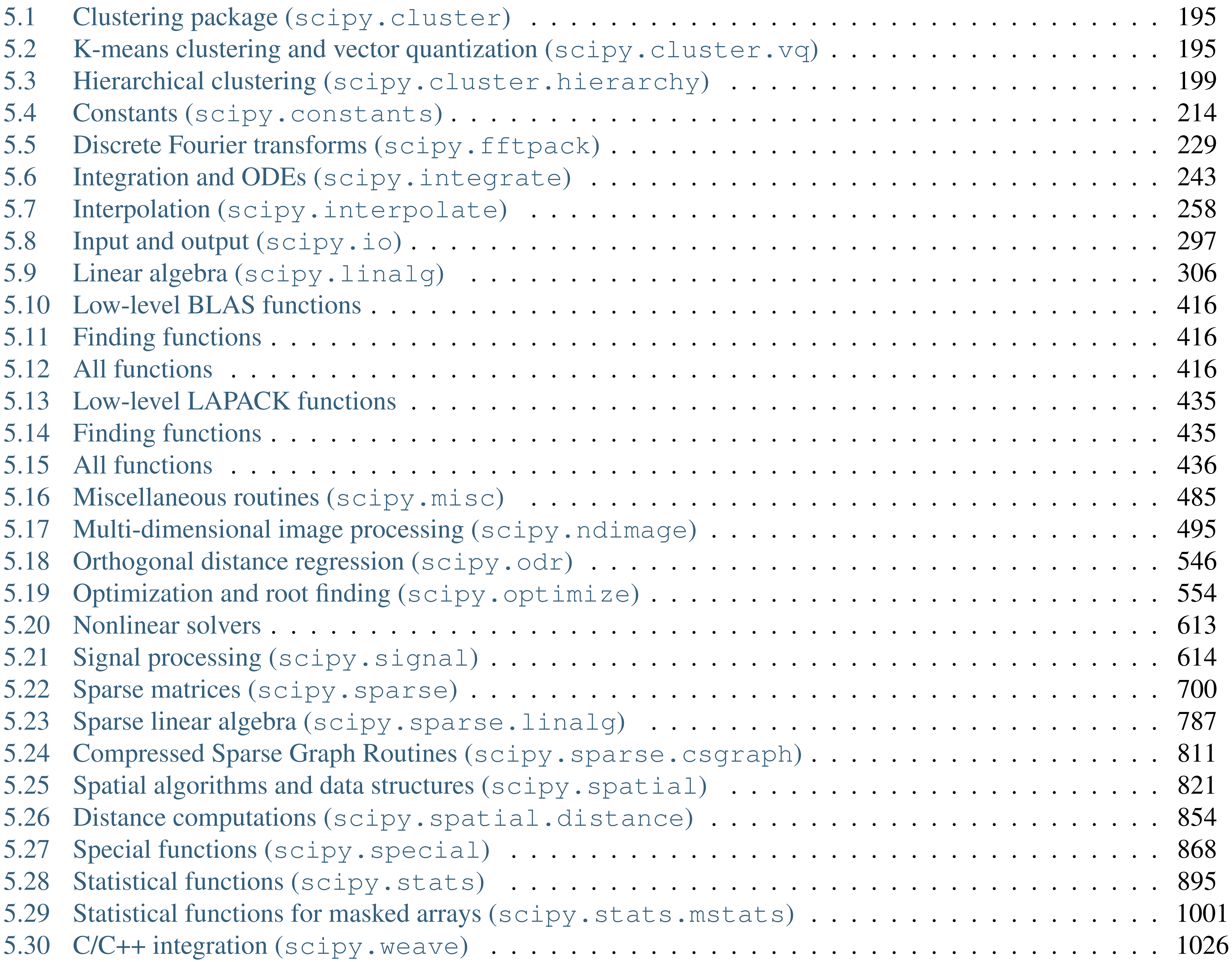
$$\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 2 & 2 & 2 \\ \hline 3 & 3 & 3 \\ \hline \end{array} - \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline 1 & 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 0 & -1 & -2 \\ \hline 1 & 0 & -1 \\ \hline 2 & 1 & 0 \\ \hline \end{array}$$

Introduction to NumPy

- Conditions and advanced indexing

```
>>> arange(8)
array([0, 1, 2, 3, 4, 5, 6, 7])

>>> d = arange(8).reshape((2,4))
>>> d
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])

>>> i = (d%2==0)
>>> i
array([[ True, False,  True, False],
       [ True, False,  True, False]], dtype=bool)

>>> d[i]
array([0, 2, 4, 6])

>>> (d<0).any()
False

>>> d[eye(3,dtype=int), eye(3,dtype=int)]
array([[5, 0, 0],
       [0, 5, 0],
       [0, 0, 5]])
```

Some sub-packages of SciPy



ROS and Python

- ROS is intimately related with Python

  in fact, many command-line tools are written in Python!
- **rospy** is the main package to use ROS from Python
  - Message types map to Python classes and messages map to Python objects
  - Publishers are Python objects
  - Subscribers call Python callback functions
  - Service clients map to Python (proxy) functions
  - Service servers map to Python callback functions
  - also contain many other useful functions and classes
- Recommended deployment:

  Place executable Python scripts in the scripts/ dir. in package

Writing a publisher

- rospy.Publisher(<topic name>, <message class>)
- rospy.init_node(<node name>)
- <publisher object>.publish(<message object>)

```
#! /usr/bin/env python

import rospy
from std_msgs.msg import *

def main():
    n = 0
    pub = rospy.Publisher("abc", String)
    rospy.init_node("publisher")
    while not rospy.is_shutdown():
        data = "hello world #%s"%(n)
        n += 1
        pub.publish(data)
        rospy.sleep(1)

if __name__=="__main__":
    main()
```

Writing a subscriber

- rospy.Subscriber(<topic>, <message class>, <callback>)
- rospy.spin()
- <callback>(<message object>)

```
#! /usr/bin/env python

import rospy
from std_msgs.msg import *

def callback(msg):
    print "Received '%s'"%(msg.data)

def main():
    rospy.init_node("subscriber")
    rospy.Subscriber("abc", String, callback)
    rospy.spin()

if __name__=="__main__":
    main()
```

Publishing a custom message

```
# file: Abc.msg
uint32 a
string b
float64[] c
```

```
#! /usr/bin/env python

import rospy
from xpto.msg import *

def main():
    n = 0
    pub = rospy.Publisher("abc", Abc)
    rospy.init_node("publisher")
    while not rospy.is_shutdown():
        msg = Abc(a=n)
        msg.b = "hello world #%s"%(n)
        msg.c = [n, 2.0*n, n/2.0]
        n += 1
        pub.publish(msg)
        rospy.sleep(1)

if __name__=="__main__":
    main()
```

Subscribing to a custom message

```
# file: Abc.msg
uint32 a
string b
float64[] c
```

```
#! /usr/bin/env python

import rospy
from xpto.msg import *

def callback(msg):
    print "Received a=%s b=%s c=%s"%(msg.a, msg.b, msg.c)

def main():
    rospy.init_node("subscriber")
    rospy.Subscriber("abc", Abc, callback)
    rospy.spin()

if __name__=="__main__":
    main()
```

Service server

```
# file: Def.msg
float64[] data
---
float64 average
float64 stddev
```

```
#! /usr/bin/env python

import numpy as np
import rospy
from xpto.srv import *

def handler(req):
    avr = np.average(req.data)
    std = np.std(req.data)
    return DefResponse(average=avr, stddev=std)

def main():
    rospy.init_node("server")
    s = rospy.Service("adder", Def, handler)
    rospy.spin()

if __name__=="__main__":
    main()
```

Service client

```
# file: Def.msg
float64[] data
---
float64 average
float64 stddev
```

```
#! /usr/bin/env python

import rospy
from xpto.srv import *

def main():
    n = 0
    rospy.wait_for_service("adder")
    f = rospy.ServiceProxy("adder", Def)
    while not rospy.is_shutdown():
        r =  f( [n, 2.0*n, n/2.0] )
        n += 1
        print r.average, r.stddev
        rospy.sleep(1)

if __name__=="__main__":
    main()
```

Opening rosbags

- **rosbag** package reads/writes rosbag files directly

```
>>> from rosbag import *
>>> bag = Bag("mapping_1-2012-02-29-11-35-57.bag")
>>> bag
<rosbag.bag.Bag object at 0x1107bf350>

>>> i = iter(bag)
>>> i.next()
[...]
>>> i.next()
('/scout/localization',
  header:
    seq: 1
    stamp:
      secs: 1330515358
      nsecs: 848419904
    frame_id: ''
  x: 0.675562620163
  y: 6.86913537979
  theta: 1.46541321278,
 genpy.Time[1330515358848848983])
```

message

Opening rosbags

- **iteration returns** (<topic name>, <message object>, <time>)
- Example: processing rosbags

```
with Bag("mapping_1-2012-02-29-11-35-57.bag") as src:
    with Bag("output.bag", "w") as dst:
            for (topic, msg, time) in src:
                    # process message here
                    dst.write(topic, msg, t=time)
```

Other useful funtions

- **Class** rospy.Rate **to loop at a specified rate**
- **Parameter access functions:**
  - rospy.has_param(<param_name>)
  - rospy. get_param(<param_name>, [<default>])
  - rospy. set_param(<param_name>, <param_value>)
  - ...
- **Logging functions:**
  - rospy.loginfo(<msg>, [<args>]*)
  - rospy.logwarn(<msg>, [<args>]*)
  - rospy.logerr(<msg>, [<args>]*)
  - ...
- **Function** rospy.get_rostime() **to get current time**