Jonas van den Brink, jvbrink

j.v.d.brink@fys.uio.no

6.1—Find the paths to a collection of programs

```python
import os

def findprograms(programs, dirs=[]):
    '''
    Looks for the specified programs, and returns a dictionary
    containing the programs' complete path on the current system.
    PATH is searched by default and additional directories can
    be specified using the dirs parameter.
    '''
    dictionary = {}

    for program in programs:
        # Check all paths in PATH and additional directories
        for path in os.environ["PATH"].split(os.pathsep) + dirs:
            # Create the filepath for the executable
            filepath = os.path.join(path, program)
            # Check if file exists and is executable
            if os.path.isfile(filepath) and os.access(filepath, os.X_OK):
                # Program exists on computer, move on to next program
                dictionary[program] = filepath
                break
            # Program does not exist, set value to None
            dictionary[program] = None

    return dictionary



if __name__ == '__main__':
    # Example of use
    programs = {
        'gnuplot'  : 'plotting program',
        'gs'       : 'ghostscript, ps/pdf interpreter and previewer',
        'f2py'     : 'generator for Python interfaces to F77',
        'swig'     : 'generator for Python interfaces to C/C++',
        'convert'  : 'image conversion, part of the ImageMagick package',
        }

    installed = findprograms(programs.keys(), ['/usr/bin'])

    for program in installed.keys():
        if installed[program]:
           print "You have %s (%s)" % (program, programs[program])
        else:
            print "*** Program %s was not found on the system" % (program,)


'''
user$ python findprograms.py
You have convert (image conversion, part of the ImageMagick package)
You have gs (ghostscript, ps/pdf interpreter and previewer)
*** Program swig was not found on the system
You have gnuplot (plotting program)
You have f2py (generator for Python interfaces to F77)
'''
```

6.2—Find old and large files in a directory tree

6.3—Estimate the chance of an event in a dice game

```
import random, sys

# Read number of experiments from cmd-line
try:
    N = eval(sys.argv[1])
except IndexError:
    errormsg = 'IndexError: Number of experiments must be specified.';
    print errormsg; sys.exit(1)
except ValueError:
    errormsg = 'ValueError: Input must be an integer.'
    print errormsg; sys.exit(1)

# Perform experiments
counter = 0
for experiment in range(N):
    results = [random.randint(1,6) for i in range(2)]
    if 6 in results:
        counter += 1

# Calculate probability
p = counter/float(N)
p_exact = 11./36
error = abs(p - p_exact)

# Print results
print """
Number of experiments: %d
Exact probability:     %.4g
Estimated probability: %.4g
Error:                 %.2e
""" % (N, p_exact, p, error)


"""
user$ python dice2.py 1000

Number of experiments: 1000
Exact probability:     0.3056
Estimated probability: 0.304
Error:                 1.56e-03

user$ python dice2.py 100000

Number of experiments: 100000
Exact probability:     0.3056
Estimated probability: 0.3054
Error:                 1.66e-04

user$ python dice2.py 10000000

Number of experiments: 10000000
Exact probability:     0.3056
Estimated probability: 0.3055
Error:                 5.69e-05
"""
```

6.4—Determine if you win or loose a hazard game

```
import numpy as np
import sys

# Read number of games from cmd-line
try:
    N = eval(sys.argv[1])
except IndexError:
    errormsg = ’IndexError: Number of games must be specified.’;
    print errormsg; sys.exit(1)
except ValueError:
    errormsg = ’ValueError: Input must be an integer.’
    print errormsg; sys.exit(1)

# Draw 4 integers from [1,7), N times
results = np.random.randint(1,7,(4,N))
# Sum the 4 integers for every game
s = np.sum(results, axis=0)
# Separate winning and loosing results
wins = s[s<9]
loss = s[s>=9]
# Calculate losses and rewards
money = 9*len(wins) - len(loss)
average = money/float(N)
# Analyze results
r = ’No’ if average < 0 else ’Yes’

# Print results
print """
Number of games:            %d
Total money won/lost:       %g
Estimated average winning: %.2g
Should you play?:           %s
""" % (N, money, average, r)


"""
user$ python dice4.py 1000

Number of games:            1000
Total money won/lost:       -480
Estimated average winning: -0.48
Should you play?:           No

user$ python dice4.py 1000000

Number of games:            1000000
Total money won/lost:       -463230
Estimated average winning: -0.46
Should you play?:           No
"""
```

6.5—Implement a class for vectors in 3D

```
class Vec3D:
    '''Class for representing real 3D vectors'''

    def __init__(self, x, y, z):
        '''Constructor, takes the vectors coordinates'''
        self.coordinates = [x, y, z]

    def __add__(self, other):
        '''Adds two vectors together, defining a new vector'''
        x1, y1, z1 = self.coordinates
        x2, y2, z2 = other.coordinates
        return Vec3D(x1+x2, y1+y2, z1+z2)

    def __sub__(self, other):
        '''Subtracts the second vector from the first, defining a third'''
        x1, y1, z1 = self.coordinates
        x2, y2, z2 = other.coordinates
        return Vec3D(x1-x2, y1-y2, z1-z2)

    def __mul__(self, other):
        '''Vector scalar product between two vectors, returns a scalar'''
        x1, y1, z1 = self.coordinates
        x2, y2, z2 = other.coordinates
        return x1*x2 + y1*y2 + z1*z2

    def __pow__(self, other):
        '''Vector cross product between two vectors, returns a Vec3D'''
        x1, y1, z1 = self.coordinates
        x2, y2, z2 = other.coordinates
        x = y1*z2 - y2*z1
        y = x2*z1 - x1*z2
        z = x1*y2 - x2*z1
        return Vec3D(x, y, z)

    def __getitem__(self, key):
        '''Allows subscripting of the vector'''
        return self.coordinates[key]

    def __setitem__(self, key, value):
        '''Allows assignment through subscripting'''
        self.coordinates[key] = value

    def __str__(self):
        '''Informal string representation of the vector'''
        x, y, z = self.coordinates
        return '(%g, %g, %g)' % (x, y, z)

    def __repr__(self):
        '''Formal string representation of the object'''
        x, y, z = self.coordinates
        return 'Vec3D(%g, %g, %g)' % (x, y, z)

    def len(self):
        '''Returns the eucledian norm'''
        x, y, z = self.coordinates
        return (x**2 + y**2 + z**2)**(1./2)


if __name__=="__main__":
    # Example run
    u = Vec3D(1, 0, 0)
    u = eval(repr(u))  # Should leave u unchanged
    v = Vec3D(0, 1, 0)
    v[2] = 2.5
    print "str(u)  ", str(u)
    print "u.len() ", u.len()
    print "u[1]    ", u[1]
    print "print v ", v
    print "u**v    ", u**v
    print "u+v     ", u+v
    print "u-v     ", u-v
    print "u*v     ", u*v
```

```
'''
user$ python vec3d.py
str(u)   (1, 0, 0)
u.len()  1.0
u[1]     0
print v  (0, 1, 2.5)
u**v     (0, -2.5, 1)
u+v      (1, 1, 2.5)
u-v      (1, -1, -2.5)
u*v      0.0
'''
```