Using Functions in Other Programs

Using modular programming we can not only divide a program into functions, but also keep them in different files.

We distinguish between two types of Python programs:
- A module contains functions that are available for use by other programs.
- A client is a program that makes use of a function in a module.

Five steps involved in creating and using modules:
1. In the client, import the module.
2. In the client, qualify function calls to the module.
3. In the module, compose a test client.
4. In the module, eliminate arbitrary global code.
5. Make the module accessible to the client.

Modular programming enables us to independently develop and debug functions for an application and then utilize them at any later time.

Modular Programming Abstractions

User-defined modules are files that each contain a set of related functions for use by other programs.

We use three abstractions to manage the process of developing user-defined modules:
1. Application programming interfaces (APIs)
   - pdf(x, mu, sigma)
   - cdf(z, mu, sigma)

2. Clients
   - percent = gaussian.cdf(score, mu, sigma)

3. Implementations
   - def pdf(x, mu = 0.0, sigma = 1.0):
   - def cdf(z, mu = 0.0, sigma = 1.0):
Modular Programming Abstractions
A library is a collection of related modules
For example, NumPy is a library for scientific computing
A private function, having its name start with an underscore by convention, is a helper
def _phi (x):
    return math . exp (- x * x / 2.0) / math . sqrt (2 * math .pi)
def phi (x, mu = 0.0 , sigma = 1.0):
    return _phi (float ((x - mu) / sigma )) / sigma

Documentation for modules and their functions is provided by embedding the
documentation string in triple quotes

""
stdrandom .py
The stdrandom module defines functions related to pseudo - random numbers .
""

... def uniformInt (lo , hi ):
    ""
    Return an integer chosen uniformly from the range [lo , hi ).
    ""
    return random . randrange (lo , hi)
...

Random Numbers

API for stdrandom module

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>seed(i)</td>
<td>seed the random number generator with integer i</td>
</tr>
<tr>
<td>uniformInt(lo, hi)</td>
<td>uniform random integer in the range [lo, hi]</td>
</tr>
<tr>
<td>bernoulli(p)</td>
<td>True with probability p (defaults to 0.5)</td>
</tr>
<tr>
<td>binomial(n, p)</td>
<td>number of heads in n coin flips, each of which is heads with probability p (defaults to 0.5)</td>
</tr>
<tr>
<td>gaussian(mu, sigma)</td>
<td>normal, mean mu (defaults to 0), standard deviation sigma (defaults to 1)</td>
</tr>
<tr>
<td>discrete(a)</td>
<td>i with probability proportional to a[i]</td>
</tr>
<tr>
<td>shuffle(a)</td>
<td>randomly shuffle the list a</td>
</tr>
<tr>
<td>exp(lambd)</td>
<td>a float from an exponential distribution with rate lambd</td>
</tr>
</tbody>
</table>

Random Numbers

stdrandom.client.py: Accept integer trials as a command-line argument. Plot trials number of (x, y) points to standard draw, where x and y are drawn from a Gaussian distribution.

```
import stddraw
import stdrandom
import sys

trials = int ( sys . argv [1])
stddraw . setPenRadius (0.0)
for i in range (trials):
    x = stdrandom . gaussian (0.5 , 0.2)
    y = stdrandom . gaussian (0.5 , 0.2)
    stddraw . point (x, y)
stddraw . show ()
```

$ python3 stdrandom_client .py 100000

Random Numbers

stdrandom.py: Random number module that defines functions related to pseudo-random numbers.

```
import random
import math

def seed(i = None):
    random . seed (i)
def uniformInt(lo, hi):
    return random . randrange (lo , hi)
def uniformFloat(lo, hi):
    return random . uniform (lo , hi)
def bernoulli(p):
    return random . random () < p
def binomial(n, p = 0.5):
    heads = 0
    for i in range(n):
        if bernoulli(p):
            heads += 1
    return heads
def gaussian(mean = 0.0, stddev = 1.0):
    return random . gauss (mu , sigma)
```

$ python3 stdrandom.py 100000
**Random Numbers**

```python
def discrete(a):
    r = uniformFloat(0.0, sum(a))
    subtotal = 0.0
    for i in range(len(a)):
        subtotal += a[i]
        if subtotal > r:
            return i

def shuffle(a):
    random.shuffle(a)

def exp(lambd):
    return -math.log(1 - random.random()) / lambd
```

**List Processing**

**API for stdarray module**

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create1D(n, val)</td>
<td>list of length n, each element initialized to val</td>
</tr>
<tr>
<td>create2D(m, n, val)</td>
<td>m-by-n list, each element initialized to val</td>
</tr>
<tr>
<td>write1D(a)</td>
<td>write list a to standard output</td>
</tr>
<tr>
<td>write2D(a)</td>
<td>write two-dimensional list a to standard output</td>
</tr>
<tr>
<td>readInt1D()</td>
<td>list of integers, read from standard input</td>
</tr>
<tr>
<td>readInt2D()</td>
<td>two-dimensional list of integers, read from standard input</td>
</tr>
<tr>
<td>readFloat1D()</td>
<td>list of floats, read from standard input</td>
</tr>
<tr>
<td>readFloat2D()</td>
<td>two-dimensional list of floats, read from standard input</td>
</tr>
<tr>
<td>readBool1D()</td>
<td>list of booleans, read from standard input</td>
</tr>
<tr>
<td>readBool2D()</td>
<td>two-dimensional list of booleans, read from standard input</td>
</tr>
</tbody>
</table>

**List Processing**

```python
# ifs.py: Accept integer n as a command-line argument. Read a 1-by-m vector (probabilities) and two m-by-3 matrices (coefficients for updating x and y, respectively) from standard input. Plot the results as a set of n points to standard draw.

import stdarray
import stddraw
import stdrandom
import sys

def main():
    n = int(sys.argv[1])
    dist = stdarray.readFloat1D()
    cx = stdarray.readFloat2D()
    cy = stdarray.readFloat2D()
    x = 0.0
    y = 0.0
    stddraw.setPenRadius(0.0)
    for i in range(n):
        r = stdrandom.discrete(dist)
        x0 = cx[r][0] * x + cx[r][1] * y + cx[r][2]
        y0 = cy[r][0] * x + cy[r][1] * y + cy[r][2]
        x = x0
        y = y0
        stddraw.point(x, y)
    stddraw.show()

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

```bash
$ more sierpinski.txt
3
.33 .33 .34
3 3
.50 .00 .00
.50 .00 .50
.50 .00 .25
3 3
.00 .50 .00
.00 .50 .00
.00 .50 .433
$ python3 ifs.py 20000 < sierpinski.txt

$ more barnsley.txt
4
0.01 0.85 0.07 0.07
4 3
0.00 0.00 0.500
0.86 0.04 0.076
0.20 -0.26 0.400
-0.15 0.28 0.575
4 3
0.00 0.16 0.000
-0.04 0.85 0.180
0.23 0.22 0.045
0.26 0.24 -0.086
$ python3 ifs.py 20000 < barnsley.txt
```
import stdio

def create1D(length, value=None):
    return [value] * length

def create2D(rowCount, colCount, value=None):
    a = [None] * rowCount
    for row in range(rowCount):
        a[row] = [value] * colCount
    return a

def write1D(a):
    length = len(a)
    stdio.writeln(length)
    for i in range(length):
        element = a[i]
        if isinstance(element, bool):
            if element == True:
                stdio.write(1)
            else:
                stdio.write(0)
        else:
            stdio.write(element)
            stdio.write(' ')
    stdio.writeln()

def write2D(a):
    rowCount = len(a)
colCount = len(a[0])
    stdio.writeln(str(rowCount) + ' ' + str(colCount))
    for row in range(rowCount):
        for col in range(colCount):
            a[row][col] = stdio.readInt()
    return a

def readInt1D():
    count = stdio.readInt()
a = create1D(count, None)
    for i in range(count):
        a[i] = stdio.readInt()
    return a

def readInt2D():
    rowCount = stdio.readInt()
colCount = stdio.readInt()
a = create2D(rowCount, colCount, 0)
    for row in range(rowCount):
        for col in range(colCount):
            a[row][col] = stdio.readInt()
    return a

def readFloat1D():
    count = stdio.readInt()
a = create1D(count, None)
    for i in range(count):
        a[i] = stdio.readFloat()
    return a

def readFloat2D():
    rowCount = stdio.readInt()
colCount = stdio.readInt()
a = create2D(rowCount, colCount, 0.0)
    for row in range(rowCount):
        for col in range(colCount):
            a[row][col] = stdio.readFloat()
    return a

def readBool1D():
    count = stdio.readInt()
a = create1D(count, None)
    for i in range(count):
        a[i] = stdio.readBool()
    return a

def readBool2D():
    rowCount = stdio.readInt()
colCount = stdio.readInt()
a = create2D(rowCount, colCount, False)
    for row in range(rowCount):
        for col in range(colCount):
            a[row][col] = stdio.readBool()
    return a

def _main():
    write2D(readFloat2D())
    write2D(readBool2D())

if __name__ == '__main__':
    _main()
API for stdstats module

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean(a)</td>
<td>average of the values in the numeric list a</td>
</tr>
<tr>
<td>var(a)</td>
<td>sample variance of the values in the numeric list a</td>
</tr>
<tr>
<td>stddev(a)</td>
<td>sample standard deviation of the values in the numeric list a</td>
</tr>
<tr>
<td>median(a)</td>
<td>median of the values in the numeric list a</td>
</tr>
<tr>
<td>plotPoints(a)</td>
<td>point plot of the values in the numeric list a</td>
</tr>
<tr>
<td>plotBars(a)</td>
<td>bar plot of the values in the numeric list a</td>
</tr>
</tbody>
</table>

import stdarray
import stddraw
import stdrandom
import stdstats
import sys

def main():
    n = int(sys.argv[1])
    trials = int(sys.argv[2])
    freq = stdarray.create1D(n + 1, 0)
    for t in range(trials):
        heads = stdrandom.binomial(n, 0.5)
        freq[heads] += 1
    norm = stdarray.create1D(n + 1, 0.0)
    for i in range(n + 1):
        norm[i] = 1.0 * freq[i] / trials
    stddev = math.sqrt(norm[n] / 2.0)
    phi = stdarray.create1D(n + 1, 0.0)
    for i in range(n + 1):
        phi[i] = gaussian.pdf(i, n / 2.0, stddev)
    stddraw.setCanvasSize(1000, 400)
    stddraw.setYscale(0, 1.1 * max(max(norm), max(phi)))
    stdstats.plotBars(norm)
    stdstats.plotLines(phi)
    stddraw.show()

if __name__ == '__main__':
    main()
def plotPoints(a):
    n = len(a)
    stddraw.setXscale(-1, n)
    stddraw.setPenRadius(1.0 / (3.0 * n))
    for i in range(n):
        stddraw.point(i, a[i])

def plotLines(a):
    n = len(a)
    stddraw.setXscale(-1, n)
    stddraw.setPenRadius(0.0)
    for i in range(1, n):
        stddraw.line(i - 1, a[i - 1], i, a[i])

def plotBars(a):
    n = len(a)
    stddraw.setXscale(-1, n)
    for i in range(n):
        stddraw.filledRectangle(i - 0.25, 0.0, 0.5, a[i])

def _main():
    import stdarray
    import stdio
    a = stdarray.readFloat1D()
    stdio.writeln('mean %7.3f\n', mean(a))
    stdio.writeln('std dev %7.3f\n', stddev(a))
    stdio.writeln('median %7.3f\n', median(a))

if __name__ == '__main__':
    _main()