Introduction to Programming in Python

Procedural Programming: Libraries and Applications

Outline

Libraries Versus Applications

Libraries Versus Applications

We distinguish between two types of Python programs

1. Libraries
2. Applications

Libraries Versus Applications

We distinguish between two types of Python programs

1. Libraries
2. Applications

Libraries are files that each contain a set of related functions for use by applications

Libraries Versus Applications

We distinguish between two types of Python programs

1. Libraries
2. Applications

Libraries are files that each contain a set of related functions for use by applications

Example: `math`, `stdarray`, `stdrandom`

Libraries Versus Applications

We distinguish between two types of Python programs

1. Libraries
2. Applications

Libraries are files that each contain a set of related functions for use by applications

Example: `math`, `stdarray`, `stdrandom`

Applications are standalone programs that are executed directly

Libraries Versus Applications

We distinguish between two types of Python programs

1. Libraries
2. Applications

Libraries are files that each contain a set of related functions for use by applications

Example: `math`, `stdarray`, `stdrandom`

Applications are standalone programs that are executed directly

Example: `quadratic.py`, `sample.py`, `gambler.py`

Libraries Versus Applications

Libraries Versus Applications

Developing a library involves

- Designing an API for the library
- Implementing the API
- Testing the library

`matrix`

`row(a, i)`	returns the ith row of matrix a
`col(a, j)`	returns the jth column of matrix a
`add(a, b)`	returns the sum of matrices a and b
`subtract(a, b)`	returns the difference of matrices a and b
`multiply(a, b)`	returns the product of matrices a and b
`transpose(a)`	returns the tranpose of matrix a
`dot(a, b)`	returns the dot-product of 1-by-n matrices a and b

`ifs.py`

Command-line input	n (int)
Standard input	a $1 \times m$ vector of probabilities and two $m \times 3$ matrices of x, y coefficients
Standard draw output	a set of n points

```
>_ ~/workspace/ipp/programs

$ _
```

```
>_ ~/workspace/ipp/programs

$ cat data/sierpinksi.txt
```

```
>_ ~/workspace/ipp/programs

$ cat data/sierpinksi.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
$ _
```

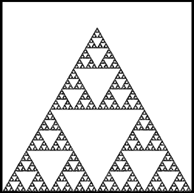
```
>_ ~/workspace/ipp/programs

$ cat data/sierpinksi.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 100000 < data/sierpinski.txt
```

Library of Matrix Functions · Using the Library

```
>_ ~/workspace/ipp/programs
```

```
$ cat data/sierpinksi.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 100000 < data/sierpinski.txt
```



```
>_ ~/workspace/ipp/programs

$ cat data/sierpinksi.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 100000 < data/sierpinski.txt
$ _
```

```
>_ ~/workspace/ipp/programs

$ _
```

```
>_ ~/workspace/ipp/programs

$ cat data/barnsley.txt
```

```
>_ ~/workspace/ipp/programs

$ cat data/barnsley.txt
4
   0.01  0.85  0.07  0.07
4 3
   0.00  0.00  0.500
   0.85  0.04  0.075
   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
   0.23  0.22  0.045
   0.26  0.24 -0.086
$ _
```

```
>_ ~/workspace/ipp/programs

$ cat data/barnsley.txt
4
   0.01  0.85  0.07  0.07
4 3
   0.00  0.00  0.500
   0.85  0.04  0.075
   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
   0.23  0.22  0.045
   0.26  0.24 -0.086
$ python3 ifs.py 100000 < data/barnsley.txt
```

```
>_ ~/workspace/ipp/programs

$ cat data/barnsley.txt
4
   0.01  0.85  0.07  0.07
4 3
   0.00  0.00  0.500
   0.85  0.04  0.075
   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
   0.23  0.22  0.045
   0.26  0.24 -0.086
$ python3 ifs.py 100000 < data/barnsley.txt
```

```
>_ ~/workspace/ipp/programs

$ cat data/barnsley.txt
4
   0.01  0.85  0.07  0.07
4 3
   0.00  0.00  0.500
   0.85  0.04  0.075
   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
   0.23  0.22  0.045
   0.26  0.24 -0.086
$ python3 ifs.py 100000 < data/barnsley.txt
$ _
```

</> ifs.py

```python
import matrix
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, y = 0.0, 0.0
    stddraw.setPenRadius(0.0)
    for i in range(n):
        r = stdrandom.discrete(dist)
        col = [x, y, 1]
        x0 = matrix.dot(matrix.row(cx, r), col)
        y0 = matrix.dot(matrix.row(cy, r), col)
        x = x0
        y = y0
        stddraw.point(x, y)
    stddraw.show()

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

matrix.py 1/2

```
import stdarray
import stdio

def row(a, i):
    return a[i]

def col(a, j):
    c = []
    for row in a:
        c += [row[j]]
    return c

def add(a, b):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = a[i][j] + b[i][j]
    return c

def subtract(a, b):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = a[i][j] - b[i][j]
    return c

def multiply(a, b):
    m, n = len(a), len(b[0])
    c = stdarray.create2D(m, n, 0.0)
    for i in range(m):
        for j in range(n):
            c[i][j] = dot(row(a, i), col(b, j))
    return c
```

matrix.py 2/2

```
def transpose(a):
    m, n = len(a), len(a[0])
    c = stdarray.create2D(n, m, 0.0)
    for i in range(m):
        for j in range(n):
            c[j][i] = a[i][j]
    return c

def dot(a, b):
    total = 0.0
    for x, y in zip(a, b):
        total += x * y
    return total

def _main():
    # Unit tests the library.

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

Library of Gaussian Functions

Library of Gaussian Functions

Gaussian probability density function (pdf) with mean 0 and standard deviation 1

$$\phi(z) = \frac{e^{-\frac{z^2}{2}}}{\sqrt{2\pi}}$$

Gaussian pdf with mean μ and standard deviation σ

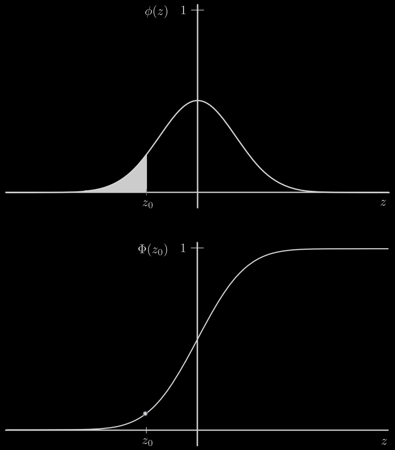
$$\phi(x, \mu, \sigma) = \frac{\phi\left(\frac{x-\mu}{\sigma}\right)}{\sigma}$$

Gaussian cumulative distribution function (cdf) with mean 0 and standard deviation 1

$$\Phi(z) = \frac{1}{2} + \phi(z)\left(z + \frac{z^3}{3} + \frac{z^5}{3 \cdot 5} + \frac{z^7}{3 \cdot 5 \cdot 7} + \cdots\right)$$

Gaussian cdf with mean μ and standard deviation σ

$$\Phi(x, \mu, \sigma) = \Phi\left(\frac{x-\mu}{\sigma}\right)$$



gaussian	
`pdf(x, mu=0.0, sigma=1.0)`	returns the value of the Gaussian pdf with mean *mu* and standard deviation *sigma* at the given x value
`cdf(x, mu=0.0, sigma=1.0)`	returns the value of the Gaussian cdf with mean *mu* and standard deviation *sigma* at the given x value

gaussiantable.py

Command-line input	μ (float) and σ (float)
Standard output	a table of the percentage of students scoring below certain scores on the SAT

gaussiantable.py

Command-line input	μ (float) and σ (float)
Standard output	a table of the percentage of students scoring below certain scores on the SAT

~/workspace/ipp/programs

```
$ _
```

gaussiantable.py

Command-line input	μ (float) and σ (float)
Standard output	a table of the percentage of students scoring below certain scores on the SAT

~/workspace/ipp/programs

```
$ python3 gaussiantable.py 1019 209
```

gaussiantable.py

Command-line input	μ (float) and σ (float)
Standard output	a table of the percentage of students scoring below certain scores on the SAT

~/workspace/ipp/programs

```
$ python3 gaussiantable.py 1019 209
 400  0.0015
 500  0.0065
 600  0.0225
 700  0.0635
 800  0.1474
 900  0.2845
1000  0.4638
1100  0.6508
1200  0.8068
1300  0.9106
1400  0.9658
1500  0.9893
1600  0.9973
$ _
```

gaussiantable.py

```
import gaussian
import stdio
import sys

def main():
    mu = float(sys.argv[1])
    sigma = float(sys.argv[2])
    for score in range(400, 1600 + 1, 100):
        percentile = gaussian.cdf(score, mu, sigma)
        stdio.writef("%4d  %.4f\n", score, percentile)

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

gaussian.py

```
import math
import stdio
import sys

def pdf(x, mu=0.0, sigma=1.0):
    z = (x - mu) / sigma
    return _pdf(z) / sigma

def cdf(x, mu=0.0, sigma=1.0):
    z = (x - mu) / sigma
    return _cdf(z)

def _pdf(z):
    return math.exp(-z * z / 2) / math.sqrt(2 * math.pi)

def _cdf(z):
    if z < -8.0:
        return 0.0
    if z > +8.0:
        return 1.0
    total = 0.0
    term = z
    i = 3
    while total != total + term:
        total += term
        term *= z * z / i
        i += 2
    return 1 / 2 + total * _pdf(z)

def _main():
    # Unit tests the library.

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