Libraries and Applications
Outline

1 Libraries and Applications

2 Gaussian Functions

3 Matrix Functions
Libraries and 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.

Developing a library involves:
• Designing an API for the library
• Implementing the API
We distinguish between two types of Python programs:

1. Libraries are files that each contain a set of related functions for use by applications. Examples: `math`, `stdarray`, `stdrandom`.
2. Applications are standalone programs that are executed directly. Examples: `quadratic.py`, `sample.py`, `gambler.py`.

Developing a library involves:

- Designing an API for the library
- Implementing the API
Libraries and Applications

We distinguish between two types of Python programs:

1. Libraries

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

2. Applications

   Example: `quadratic.py`, `sample.py`, `gambler.py`
We distinguish between two types of Python programs:

1. Libraries
2. 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.
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
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.
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 and 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

Developing a library involves:
Libraries and 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

Developing a library involves:

- Designing an API for the library
Libraries and 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`

Developing a library involves:

- Designing an API for the library
- Implementing the API
Gaussian Functions

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

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

Gaussian pdf with mean $\mu$ and standard deviation $\sigma$

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

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

$$\Phi(z) = \frac{1}{2} + \frac{\phi(z)}{\sqrt{2\pi}}$$

Gaussian cdf with mean $\mu$ and standard deviation $\sigma$

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

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

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 \( \mu \) and standard deviation \( \sigma \)

\[ \Phi(x, \mu, \sigma) = \Phi\left(\frac{x-\mu}{\sigma}\right) \]
Gaussian Functions

\texttt{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.

\texttt{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.
### Gaussian Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pdf(x, mu=0.0, sigma=1.0)</code></td>
<td>returns the value of the Gaussian pdf with mean $mu$ and standard deviation $sigma$ at the given $x$ value</td>
</tr>
<tr>
<td><code>cdf(x, mu=0.0, sigma=1.0)</code></td>
<td>returns the value of the Gaussian cdf with mean $mu$ and standard deviation $sigma$ at the given $x$ value</td>
</tr>
</tbody>
</table>
Gaussian Functions

Program: gaussiantable.py

• Command-line input: mu (float) and sigma (float)
• Standard output: a table of the percentage of students scoring below certain scores on the SAT
**Gaussian Functions**

**Program:** gaussiantable.py
Gaussian Functions

Program: gaussiantable.py

- Command-line input: $mu$ (float) and $sigma$ (float)
Gaussian Functions

Program: gaussiantable.py

- Command-line input: $mu$ (float) and $sigma$ (float)
- Standard output: a table of the percentage of students scoring below certain scores on the SAT
Gaussian Functions

Program: gaussiantable.py
- Command-line input: mu (float) and sigma (float)
- Standard output: a table of the percentage of students scoring below certain scores on the SAT
Gaussian Functions

Program: gaussiantable.py

- Command-line input: $mu$ (float) and $sigma$ (float)
- Standard output: a table of the percentage of students scoring below certain scores on the SAT

```bash
$ python3 gaussiantable.py 1019 209
```
Gaussian Functions

Program: gaussiantable.py

- Command-line input: \textit{mu} (float) and \textit{sigma} (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
$  
```
Gaussian Functions

```python
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
', score, percentile)

if __name__ == '__main__':
    main()
```
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.writeln('%4d %.4f
', score, percentile)

if __name__ == '__main__':
    main()
Gaussian Functions

```python
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 = float(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 0.5 + total * _pdf(z)

def _main():
    x = float(sys.argv[1])
    mu = float(sys.argv[2])
    sigma = float(sys.argv[3])
    stdio.writeln(cdf(x, mu, sigma))

if __name__ == '__main__':
    _main()
```
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 = float(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 0.5 + total * _pdf(z)

def _main ():
    x = float(sys.argv[1])
    mu = float(sys.argv[2])
    sigma = float(sys.argv[3])
    stdio.writeln(cdf(x, mu, sigma))
if __name__ == '__main__':
    _main()
Matrix Functions

- `row(a, i)`: returns the $i$th row of matrix $a$.
- `col(a, j)`: returns the $j$th 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 transpose of matrix $a$.
- `dot(a, b)`: returns the dot-product of 1-by-$n$ matrices $a$ and $b$. 
## Matrix Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>row(a, i)</td>
<td>returns the $i$th row of matrix $a$</td>
</tr>
<tr>
<td>col(a, j)</td>
<td>returns the $j$th column of matrix $a$</td>
</tr>
<tr>
<td>add(a, b)</td>
<td>returns the sum of matrices $a$ and $b$</td>
</tr>
<tr>
<td>subtract(a, b)</td>
<td>returns the difference of matrices $a$ and $b$</td>
</tr>
<tr>
<td>multiply(a, b)</td>
<td>returns the product of matrices $a$ and $b$</td>
</tr>
<tr>
<td>transpose(a)</td>
<td>returns the transpose of matrix $a$</td>
</tr>
<tr>
<td>dot(a, b)</td>
<td>returns the dot-product of 1-by-$n$ matrices $a$ and $b$</td>
</tr>
</tbody>
</table>
Matrix Functions

Program: ifs.py

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

\([p_0, p_1, \ldots, p_{m-1}]\),

\[
\begin{pmatrix}
x_0, 0 & x_0, 1 & x_0, 2 \\
x_1, 0 & x_1, 1 & x_1, 2 \\
\vdots & \vdots & \vdots \\
x_{m-1}, 0 & x_{m-1}, 1 & x_{m-1}, 2 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
y_0, 0 & y_0, 1 & y_0, 2 \\
y_1, 0 & y_1, 1 & y_1, 2 \\
\vdots & \vdots & \vdots \\
y_{m-1}, 0 & y_{m-1}, 1 & y_{m-1}, 2 \\
\end{pmatrix}
\]

\( r \) is an index \( i \in [0, m-1] \) from \( P \), selected with probability \( p_i \)

\( x = X_r \); \( y = Y_r \)
Matrix Functions

Program: ifs.py

r is an index $i \in [0, m-1]$ from $P$, selected with probability $p_i$:

$x = X_r$:  [$x_0, y_0, x_0, y_1, x_0, y_2, ...$]  
$y = Y_r$:  [$y_0, y_0, y_1, y_1, y_1, y_2, ...$]
Matrix Functions

Program: ifs.py

• Command-line input: $n$ (int)
Program: *ifs.py*

- Command-line input: *n* (int)
- Standard input: 1-by-*m* vector (probabilities) and two *m*-by-3 matrices (coefficients for updating *x* and *y*, respectively)

\[ P = [p_0 \ p_1 \ldots \ p_{m-1}] \]
\[ X = \begin{bmatrix} x_0,0 & x_0,1 & x_0,2 & \ldots & x_0,m-1 \\ x_1,0 & x_1,1 & x_1,2 & \ldots & x_1,m-1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m-1},0 & x_{m-1},1 & x_{m-1},2 & \ldots & x_{m-1},m-1 \end{bmatrix} \]
\[ Y = \begin{bmatrix} y_0,0 & y_0,1 & y_0,2 & \ldots & y_0,m-1 \\ y_1,0 & y_1,1 & y_1,2 & \ldots & y_1,m-1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_{m-1},0 & y_{m-1},1 & y_{m-1},2 & \ldots & y_{m-1},m-1 \end{bmatrix} \]

\[ r \text{ is an index } i \in [0, m-1] \text{ from } P, \] selected with probability \( p_i \)
\[ x = X_r \]
\[ y = Y_r \]
Program: ifs.py

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

\[
\begin{align*}
P &= [p_0, p_1, \ldots, p_{m-1}] \\
X &= \begin{bmatrix} x_0, 0 & x_0, 1 & x_0, 2 & \cdots & \cdots & \cdots & \cdots & x_m, 0 \end{bmatrix} \\
Y &= \begin{bmatrix} y_0, 0 & y_0, 1 & y_0, 2 & \cdots & \cdots & \cdots & \cdots & y_m, 0 \end{bmatrix}
\end{align*}
\]
Matrix Functions

Program: ifs.py

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

$$ P = \begin{bmatrix} p_0 & p_1 & \ldots & p_{m-1} \end{bmatrix}, \quad X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ \vdots & \vdots & \vdots \\ x_{m0} & x_{m-1,1} & x_{m-1,2} \end{bmatrix}, \quad Y = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} \\ y_{1,0} & y_{1,1} & y_{1,2} \\ \vdots & \vdots & \vdots \\ y_{m0} & y_{m-1,1} & y_{m-1,2} \end{bmatrix} $$
Matrix Functions

Program: *ifs.py*

- Command-line input: *n* (int)
- Standard input: 1-by-*m* vector (probabilities) and two *m*-by-3 matrices (coefficients for updating *x* and *y*, respectively)
- Standard draw output: a set of *n* points

\[ P = \begin{bmatrix} p_0 & p_1 & \cdots & p_{m-1} \end{bmatrix}, \quad X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ \vdots & \vdots & \vdots \\ x_{m,0} & x_{m-1,1} & x_{m-1,2} \end{bmatrix}, \quad Y = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} \\ y_{1,0} & y_{1,1} & y_{1,2} \\ \vdots & \vdots & \vdots \\ y_{m,0} & y_{m-1,1} & y_{m-1,2} \end{bmatrix} \]

*r* is an index *i* ∈ [0, *m* − 1] from *P*, selected with probability *p_i*.
Matrix Functions

Program: *ifs.py*

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

$$P = \begin{bmatrix} p_0 & p_1 & \ldots & p_{m-1} \end{bmatrix}, \quad X = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ \vdots & \vdots & \vdots \\ x_{m_0} & x_{m-1,1} & x_{m-1,2} \end{bmatrix}, \quad Y = \begin{bmatrix} y_{0,0} & y_{0,1} & y_{0,2} \\ y_{1,0} & y_{1,1} & y_{1,2} \\ \vdots & \vdots & \vdots \\ y_{m_0} & y_{m-1,1} & y_{m-1,2} \end{bmatrix}$$

$r$ is an index $i \in [0, m - 1]$ from $P$, selected with probability $p_i$

$$x = X_r \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}, \quad y = Y_r \cdot \begin{bmatrix} x & y & 1 \end{bmatrix}$$
Matrix Functions
Matrix Functions

>`~/workspace/ipp/programs`

```bash
$ cat ../data/sierpinski.txt
```
Matrix Functions

```
$ cat ../data/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
$ _
```
Matrix Functions

```bash
> ~/workspace/ipp/programs

$ cat ../data/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 < ../data/sierpinski.txt
```
Matrix Functions

```bash
$ cat ../data/sierpinski.txt
3
0.33 0.33 0.34
3 3
0.5 0.0 0.00
0.5 0.0 0.50
0.5 0.0 0.25
3 3
0.0 0.5 0.00
0.0 0.5 0.00
0.0 0.5 0.433
$ python3 ifs.py 20000 < ../data/sierpinski.txt
$ _
```
Matrix Functions
Matrix Functions

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>~/workspace/ipp/programs</code></td>
</tr>
<tr>
<td><code>$</code></td>
</tr>
</tbody>
</table>
Matrix Functions

>_ "~/workspace/ipp/programs"

$ cat ../data/barnsley.txt
<table>
<thead>
<tr>
<th>Matrix Functions</th>
</tr>
</thead>
</table>

```bash
$ 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
  0.26  0.24 -0.086
$
```

```bash
>
```
```
Matrix Functions

```bash
> ~/workspace/ipp/programs

$ cat ../data/barnsley.txt

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0.01 0.85 0.07 0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00 0.00 0.500</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.85 0.04 0.075</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.20 -0.26 0.400</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.15 0.28 0.575</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.00 0.16 0.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-0.04 0.85 0.180</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.23 0.22 0.045</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.26 0.24 -0.086</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$ python3 ifs.py 20000 < ../data/barnsley.txt
```
```
Matrix Functions

```

```
> ~/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
  0.26  0.24  -0.086
$ python3 ifs.py 20000 < ../data/barnsley.txt
$ _
```
Matrix Functions

```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()
```
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 Functions

```python
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
```
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 Functions

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 ():
    a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
    b = [[1], [2], [3]]
   stdio.writeln('a = ' + str(a))
   stdio.writeln('b = ' + str(b))
   stdio.writeln('row (a, 1) = ' + str(row(a, 1)))
   stdio.writeln('col (a, 1) = ' + str(col(a, 1)))
   stdio.writeln('subtract (a, a) = ' + str(subtract(a, a)))
   stdio.writeln('multiply (a, b) = ' + str(multiply(a, b)))
   stdio.writeln('transpose (b) = ' + str(transpose(b)))

if __name__ == '__main__':
    _main()