Outline

1. Simulation
   - Monte Carlo methods
   - random numbers

2. Repeat Until
   - binary expansion
   - break statement

3. Double For Loops
   - lists of lists to represent matrices
   - searching a list of lists

4. Summary + Assignments
Simulation
Monte Carlo methods

- In a mathematical model with uncertainties, events occur with assigned probabilities.

- Simulation consists in the repeated drawing of samples according to a probability distribution. We count the number of successful samples.

- The Law of Large Numbers states that the arithmetic average of the observed successes converges to the expected value or mean of the experiment, as the number of experiments increases.

- Monte Carlo methods are listed among the Top Ten Algorithms of the 20th century.
Running Simulations

1. Simulation
   - Monte Carlo methods
   - random numbers

2. Repeat Until
   - binary expansion
   - break statement

3. Double For Loops
   - lists of lists to represent matrices
   - searching a list of lists

4. Summary + Assignments
flowchart for simulations

```
s = 0; i = 0

? i < n?
  True
  sample; i = i + 1

? success?
  False
  print s

  True
  s = s + 1
```
Running Simulations
repeat until: break

1. Simulation
   - Monte Carlo methods
   - random numbers

2. Repeat Until
   - binary expansion
   - break statement

3. Double For Loops
   - lists of lists to represent matrices
   - searching a list of lists

4. Summary + Assignments
Random Numbers
as available in Python

Random number generators are in the module `random`. Three things we need to know:

1. `import random` loads the module into a session. Afterwards, `help(random)` shows a description of the definitions and functions offered by the module.

2. `random.seed()`
   Giving a fixed number as argument results in the same sequence of random numbers.

3. `r = random.uniform(a, b)`
   `r` is a randomly generated number, drawn from a uniform distribution over the interval \([a, b)\).
using random numbers

A sample program `randuse.py`:

```
""
Illustration of using random numbers.
""

import random # use module random
random.seed(21342342) # get same sequence
print('uniformly distributed random numbers')
LOWER = float(input('give lower bound : '))
UPPER = float(input('give upper bound : '))
RND = random.uniform(LOWER, UPPER) # generate a number
print('a random number in [%.2f, %.2f] : %.15f' % (LOWER, UPPER, RND))
```
Expected values are expressed as integrals. When many parameters are involved, the integration is high dimensional and only estimation is possible.

The area of the unit disk is $\pi$.

Generate random uniformly distributed points with coordinates $(x, y) \in [-1, +1] \times [-1, +1]$. We count a success when $x^2 + y^2 \leq 1$. 
Flowchart for Estimating $\pi$

$s = 0; i = 0$

$i < n?$

True

pick $(x, y) \in [-1, +1]^2; i = i + 1$

$x^2 + y^2 \leq 1$?

False

$s = s + 1$

print $4s/n$
estimating $\pi$ with the script mc4pi.py

"""
We count the number of samples $(x, y)$ that lie in the unit disk.
"""
from random import uniform as u

print('Monte Carlo simulation for Pi')
NBR = int(input('Give number of runs : '))
INDISK = 0
for i in range(NBR):
    (X, Y) = (u(-1, 1), u(-1, 1))
    if X**2 + Y**2 <= 1:
        INDISK = INDISK + 1
print('After %d runs : %f' % (NBR, 4*INDISK/NBR))

Why multiply by 4? 4 is the area of $[-1, +1]^2$. 

Intro to Computer Science (MCS 260) running simulations
Running Simulations
repeat until: break

1. Simulation
   - Monte Carlo methods
   - random numbers

2. Repeat Until
   - binary expansion
   - break statement

3. Double For Loops
   - lists of lists to represent matrices
   - searching a list of lists

4. Summary + Assignments
Converting Numbers

Converting 123, from decimal into binary format:

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n/2$</th>
<th>$n \mod 2$</th>
<th>$n = 61 \times 2 + 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>61</td>
<td>1</td>
<td>123 = 61 × 2 + 1</td>
</tr>
<tr>
<td>61</td>
<td>30</td>
<td>1</td>
<td>61 = 30 × 2 + 1</td>
</tr>
<tr>
<td>30</td>
<td>15</td>
<td>0</td>
<td>30 = 15 × 2 + 0</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>1</td>
<td>15 = 7 × 2 + 1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>7 = 3 × 2 + 1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3 = 1 × 2 + 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 = 0 × 2 + 1</td>
</tr>
</tbody>
</table>

123 = 1 + 2 × 61 = 1 + 2 × (1 + 2 × 30)
= 1 + 2 × (1 + 2 × (0 + 2 × 15))
= 1 + 2 × (1 + 2 × (0 + 2 × (1 + 2 × 7)))
= ... = 1111011 = 7B.

The table shows the progression of the values of the variables in the loop, each row is one iteration.
The bits of a number are the remainders of division by 2. \texttt{divmod()} is an intrinsic operation:

```python
>>> divmod(9, 2)
(4, 1)
```

Use as \((n, r) = \texttt{divmod}(n, 2)\) to obtain remainder \(n \mod 2\) in \(r\) and to replace \(n\) by \(n/2\).

Pseudocode to compute the binary expansion:

```plaintext
n = input()
repeat
    (n, r) = divmod(n, 2)
    print r
until (n == 0).
```
Flowchart of Binary Expansion

$n = \text{input}()$

$(n, r) = \text{divmod}(n, 2)$

print $r$

$n == 0? \text{True}$

$n == 0? \text{False}$
a first Python solution

"""
This first version prints the bits
in the order as they are computed.
We use divmod(), an intrinsic operation
on numeric types.
"""

print('computing the binary expansion')
NBR = int(input('Give a number : '))
(NBR, REST) = divmod(NBR, 2)
print(REST)
while NBR > 0:
    (NBR, REST) = divmod(NBR, 2)
    print(REST)

avoid duplication of code
Running Simulations
repeat until: break

1. Simulation
   - Monte Carlo methods
   - random numbers

2. Repeat Until
   - binary expansion
   - break statement

3. Double For Loops
   - lists of lists to represent matrices
   - searching a list of lists

4. Summary + Assignments
To exit a loop inside the body of a loop, the statement `break` occurs usually within an `if` statement.

```python
repeat
    < body of loop >
until < condition >
```

is realized in Python as

```python
while True:
    < body of loop >
    if < condition >:
        break
```

The `while True` starts an infinite loop, terminated when `< condition >` becomes True.
binary expansions with break: a better solution

The program below avoids the duplication of code:

"""
Use of break for repeat until.
The script also shows how to avoid a line break when printing the bits in the expansion.
"""

print('computing the binary expansion')
NBR = int(input('Give a number : '))
while True:
    (NBR, REST) = divmod(NBR, 2)
    print(REST, end='')  # no line break
if NBR == 0:
    break

Exercise: how to print bits in correct order?
Two Loops, Two Breaks – scope of a break

As long as the number typed in by the user is nonnegative, the loop continues.

"""
A break only effects one loop.
"""

print('computing the binary expansion')
while True:
    NBR = int(input('Give a number (< 0 to exit) : '))
    if NBR < 0:
        break
    while True:
        (NBR, REST) = divmod(NBR, 2)
        print(REST)
        if NBR == 0:
            break

A break only effects the one loop it is in.
Running Simulations
repeat until: break

1 Simulation
- Monte Carlo methods
- random numbers

2 Repeat Until
- binary expansion
- break statement

3 Double For Loops
- lists of lists to represent matrices
- searching a list of lists

4 Summary + Assignments
accessing lists by entry or by index

```python
>>> L = list(range(3, 10))
>>> L
[3, 4, 5, 6, 7, 8, 9]
```

We can go over the elements of `L` by entry:

```python
>>> for x in L: print(x, end=' ')
...
3 4 5 6 7 8 9 >>>
```

Or we can go over the elements of `L` by index:

```python
>>> for k in range(len(L)): print(L[k], end=' ')
...
3 4 5 6 7 8 9 >>>
```
matrices as lists of lists

We view a matrix as a list of rows.

Generating a random matrix $A$ of $N$ rows and $M$ columns:

```python
from random import randint
MAT = []
for i in range(N):
    ROW = []
    for j in range(M):
        ROW.append(randint(10, 99))
    MAT.append(ROW)
```

A typical double loop:
- $i$ runs over all the rows, from 0 to $N-1$, and
- $j$ runs over all the columns, from 0 to $M-1$. 
Running Simulations

repeat until: break

1 Simulation
- Monte Carlo methods
- random numbers

2 Repeat Until
- binary expansion
- break statement

3 Double For Loops
- lists of lists to represent matrices
- searching a list of lists

4 Summary + Assignments
searching a list of lists

Problem statement:

Input:  \( A \) is a matrix of \( n \) rows and \( m \) columns, 
        \( x \) is some number.

Output: if \( A[i][j] \) equals \( x \), then print \( [i][j] \), 
        else print \( x \) does not occur in \( A \).

We develop an interactive program:

1. The user provides \( n \) and \( m \), and

2. the computer generates an \( n \)-by-\( m \) matrix \( A \) of random integer 
   numbers in the interval \([10, +99]\).

3. The program prompts the user for \( x \), and

4. searches \( A \) for \( x \) and prints search result.
running the code at the command prompt

$ python findelem.py
give the number of rows : 3
give the number of columns : 5
random 3-by-5 matrix :
[90, 47, 93, 98, 95]
[55, 70, 51, 71, 50]
[31, 41, 23, 43, 59]
give a number : 41
found 41 at [2][1]

If the given number does not occur, then
'field does not occur in matrix' is printed.
Flowchart

\[ i = 0; \ j = 0 \]

\[ i < n? \]

\[ j < m? \]

\[ A_{i,j} = x? \]

\[ \text{print 'not found'} \]

\[ i = i + 1 \]

\[ j = j + 1 \]

\[ \text{print} (i,j) \]
Illustration of a double for loop to find an element in a two dimensional matrix.

# first we make a random matrix
from random import randint
ROWS = int(input('give the number of rows : '))
COLS = int(input('give the number of columns : '))
MAT = []
for i in range(ROWS):
    MAT.append([randint(10, 99) for _ in range(COLS)])
print('random %d-by-%d matrix :' % (ROWS, COLS))
for row in MAT:
    print(row)
the double for loop

Search an $n$-by-$m$ matrix $A$ for $x$:

```python
# then we ask for a number and search
NBR = int(input('give a number : '))
FOUND = False
for i in range(0, ROWS):
    for j in range(0, COLS):
        if MAT[i][j] == NBR:
            FOUND = True
            (ROW, COL) = (i, j)
            break
    if FOUND:
        break
```
if FOUND:
    print('found %d at [%d][%d]' % (NBR, ROW, COL))
else:
    print('%d does not occur in the matrix' % NBR)
Assignments

1. Use a stack to store the bits in the binary expansion to print the bits \textit{after} the loop in the correct order.

2. Given a list of numbers between 0 and 100, define the algorithm to assign a letter grade to each number: $\geq 90$: A, $\in [80, 89]$: B, $\in [70, 79]$: C, etc. Report at the end how many As, Bs, Cs, etc. Write the algorithm in words and draw a flowchart.

3. Implement exercise 2 in Python.

4. Write a Python program that generates $n$ numbers uniformly distributed in $[0, 1]$ and counts how many numbers are $< 0.5$.

5. Use turtle graphics to visualize the Monte Carlo method to estimate $\pi$. Represent the unit circle by a circle of radius equal to half of the width of the turtle window. Mark samples inside the disk by green circles of radius equal to 2 pixels, centered at the sample point. Use red circles for the points outside the disk.
Summary

We covered more of

- section 2.6 of *Python Programming in Context*,
- section 5.4 in *Computer Science, an overview*. 

Intro to Computer Science (MCS 260)