Outline

MCS 260 Lecture 12
Introduction to Computer Science
Jan Verschelde, 8 February 2010

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.

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.

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.

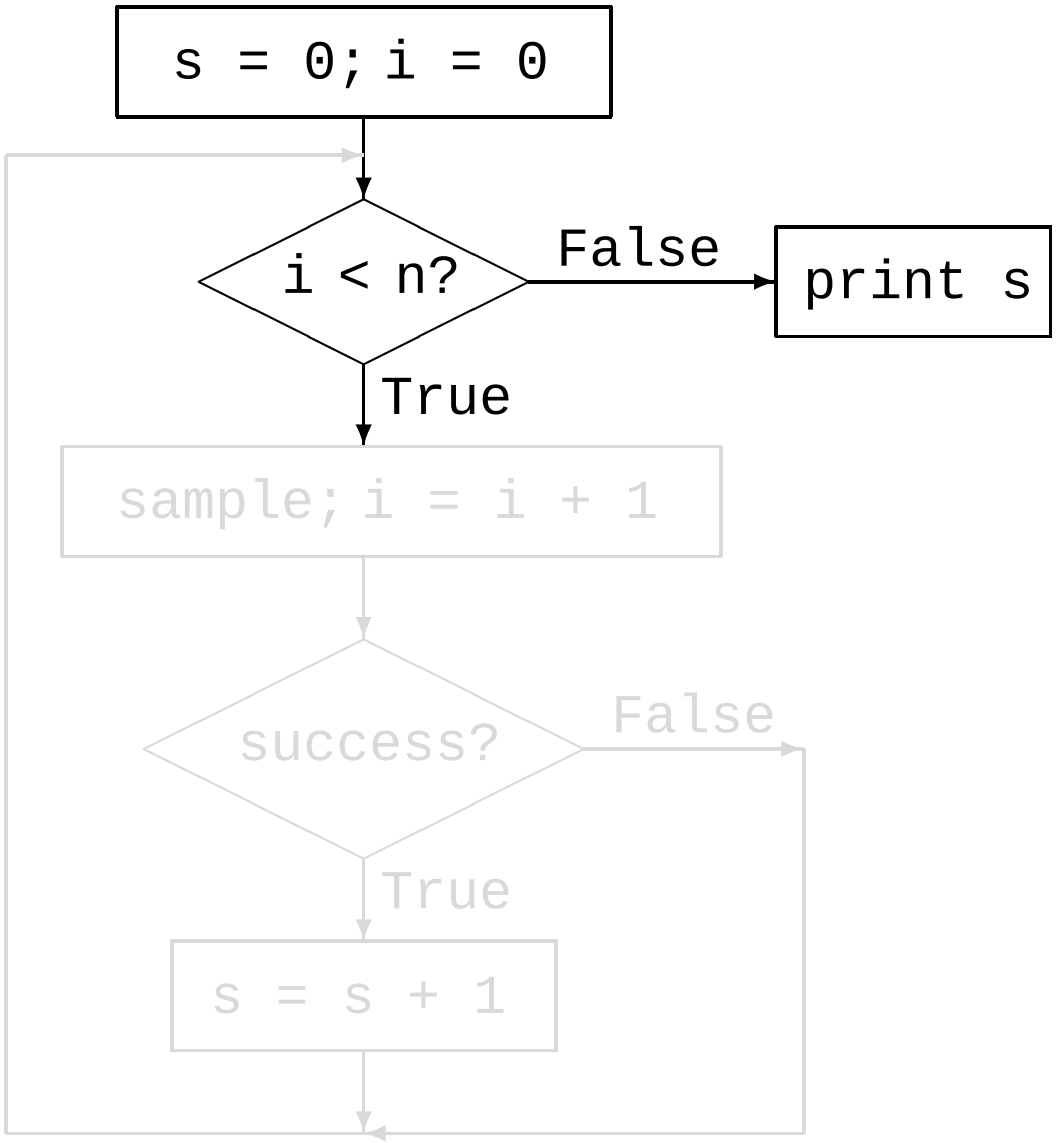
Simulation

Monte Carlo methods

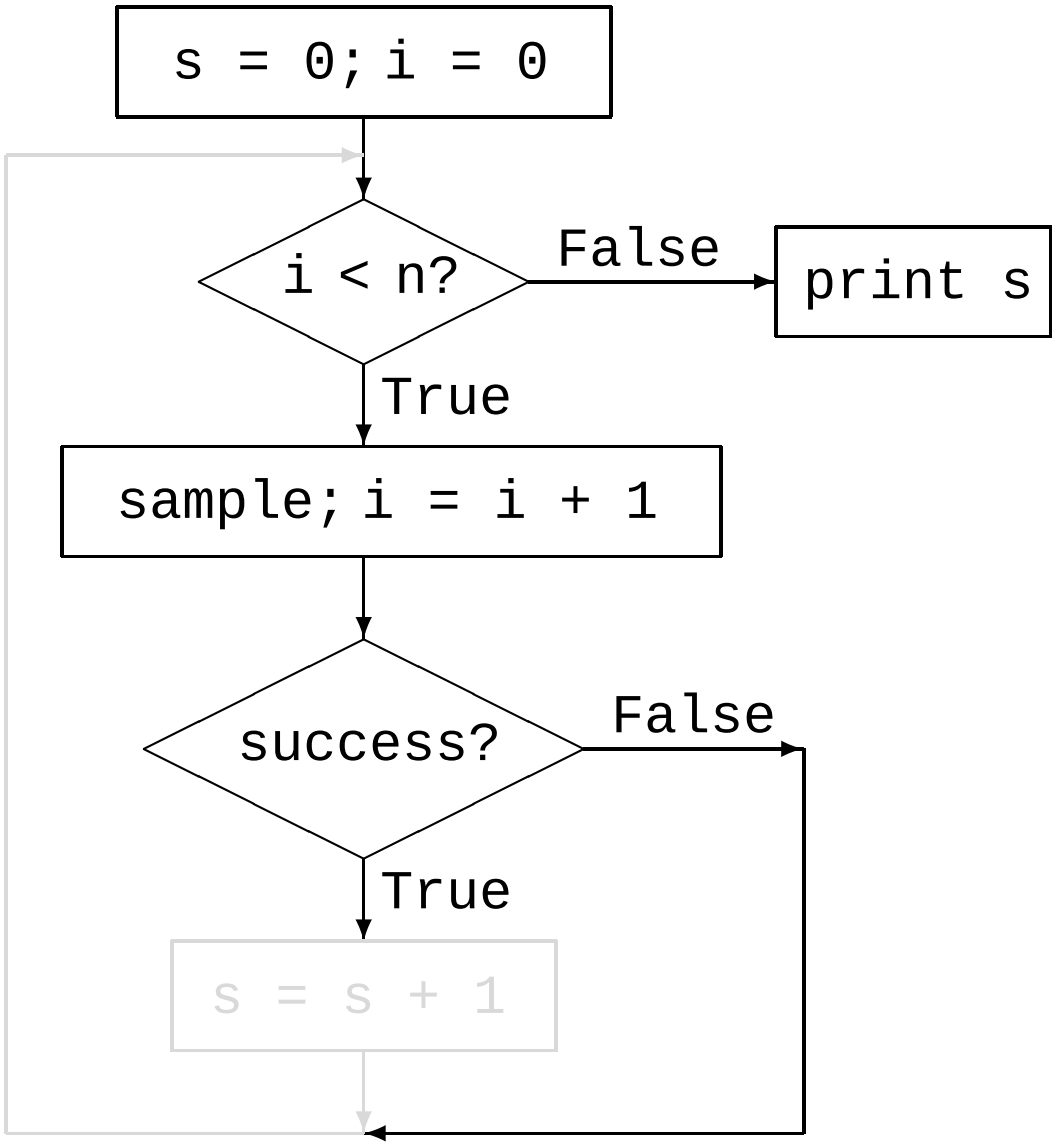
- In a mathematical model with uncertainties, events occur with assigned probabilities.
- Simulation consists in the repeated drawing of samples according to a probability distribution.
  We count the number of successful samples.
- The Law of Large Numbers states that the arithmetic average of the observed 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
repeat until: break

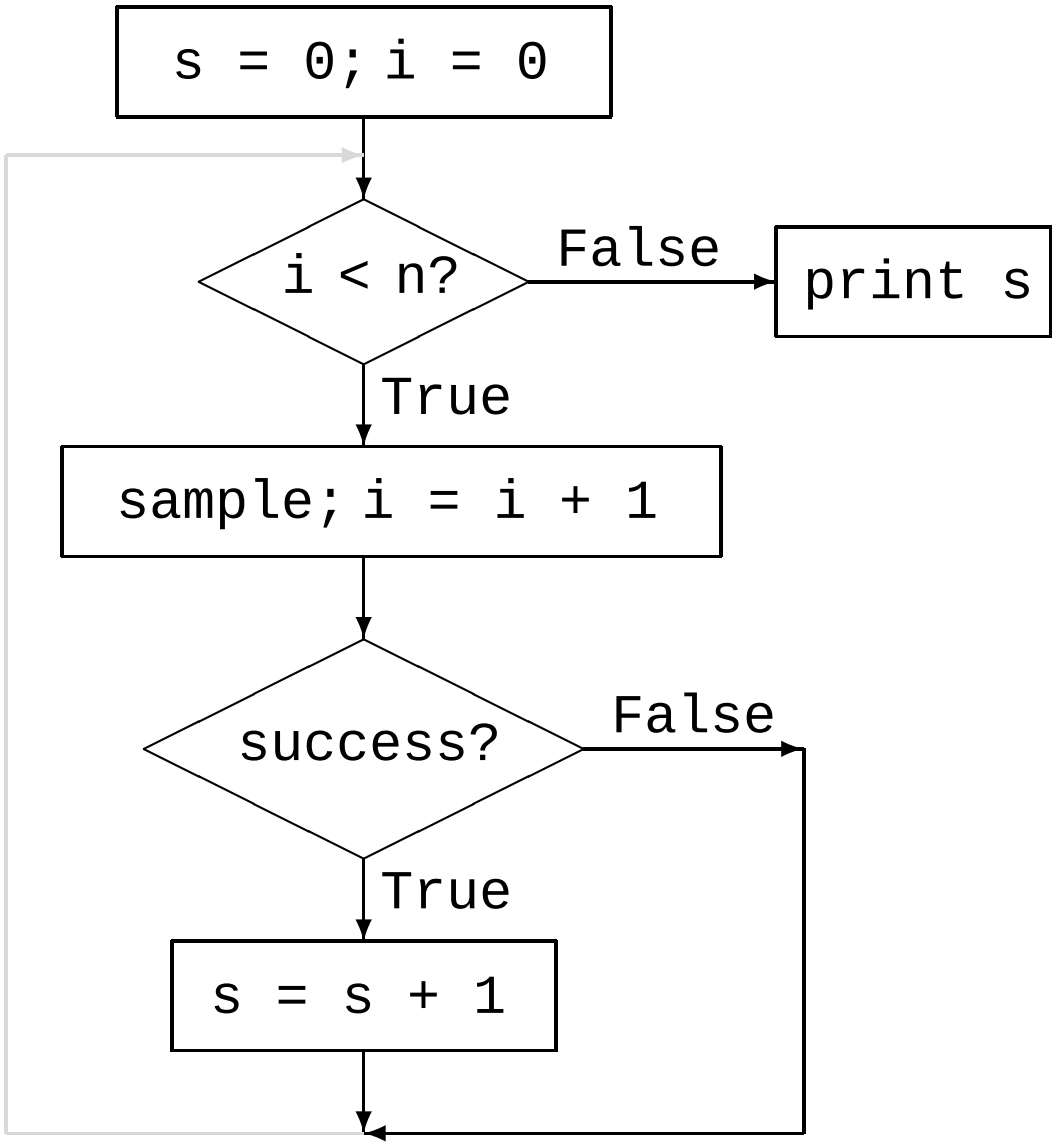
Flowchart for Simulations



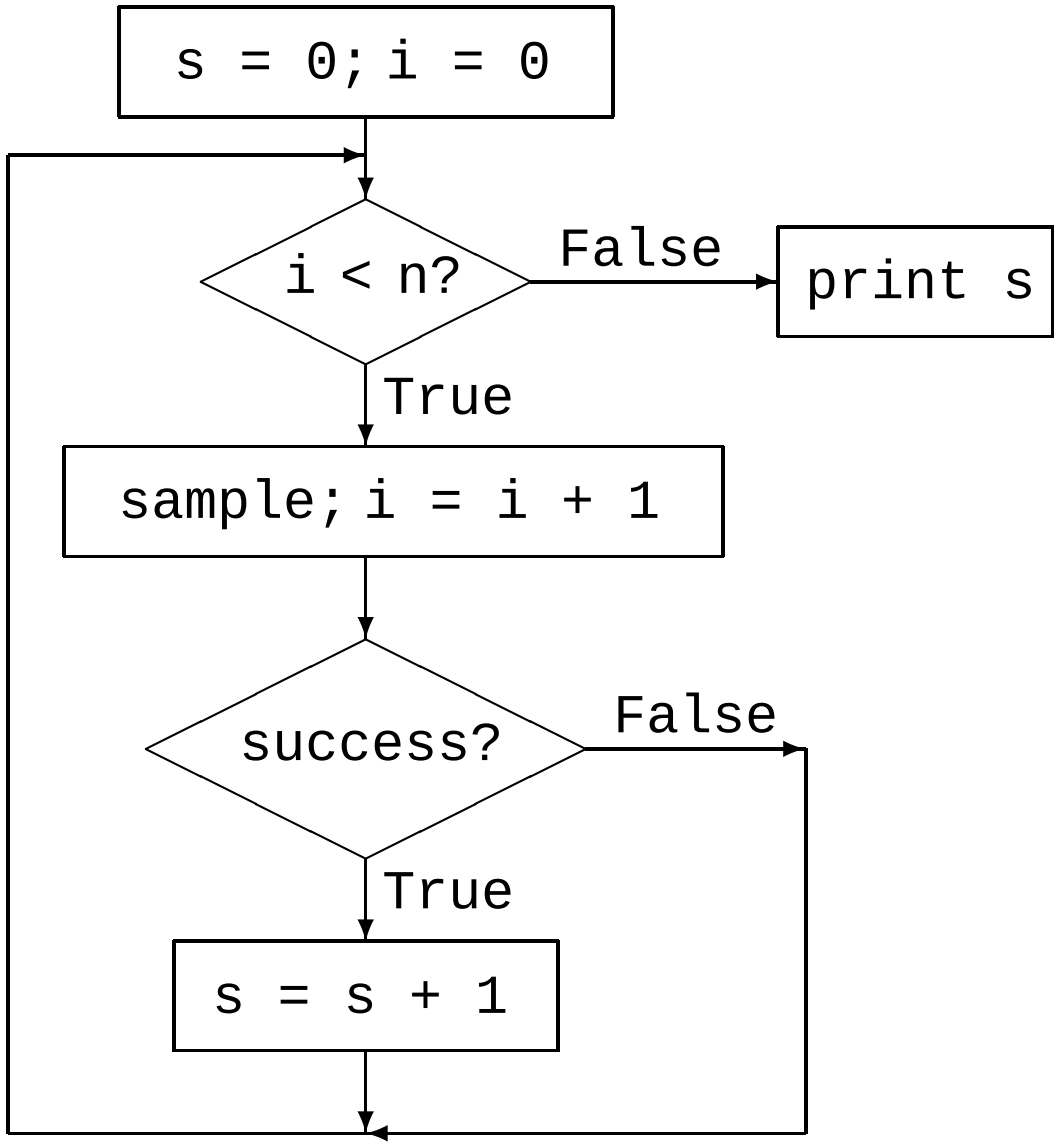
Flowchart for Simulations



Flowchart for Simulations



Flowchart for Simulations



Running Simulations
repeat until: break

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)`.

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)`.

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

A sample program randuse.py:

```
# L-12 MCS 260 Mon 8 Feb 2010  random numbers
#
import random           # use module random
random.seed(21342342) # get same sequence
print 'uniformly distributed random numbers'
a = input('give lower bound : ')
b = input('give upper bound : ')
r = random.uniform(a,b) # generate a number
print 'a random number in [%.2f,%.2f] : %.15f' \
      % (a,b,r)
```

Using Random Numbers

a sample program

A sample program randuse.py:

```
# L-12 MCS 260 Mon 8 Feb 2010  random numbers
#
import random           # use module random
random.seed(21342342) # get same sequence
print 'uniformly distributed random numbers'
a = input('give lower bound : ')
b = input('give upper bound : ')
r = random.uniform(a,b) # generate a number
print 'a random number in [%.2f,%.2f] : %.15f' \
      % (a,b,r)
```

Using Random Numbers

a sample program

A sample program randuse.py:

```
# L-12 MCS 260 Mon 8 Feb 2010  random numbers
#
import random           # use module random
random.seed(21342342) # get same sequence
print 'uniformly distributed random numbers'
a = input('give lower bound : ')
b = input('give upper bound : ')
r = random.uniform(a,b) # generate a number
print 'a random number in [%.2f,%.2f] : %.15f' \
       % (a,b,r)
```

Estimating Areas and Volumes

high dimensional integrals

- 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 π.

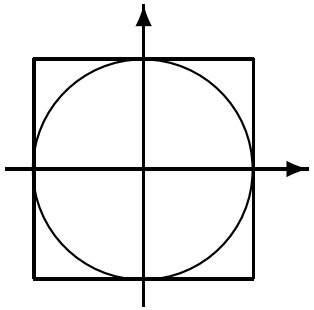
Generate random uniformly distributed points with coordinates $(x, y) \in [-1, +1] \times [-1, +1]$.

We count a success when $x^2 + y^2 \leq 1$.

Estimating Areas and Volumes

high dimensional integrals

- 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 π.



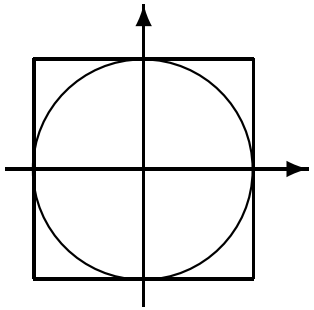
Generate random uniformly distributed points with coordinates $(x, y) \in [-1, +1] \times [-1, +1]$.

We count a success when $x^2 + y^2 \leq 1$.

Estimating Areas and Volumes

high dimensional integrals

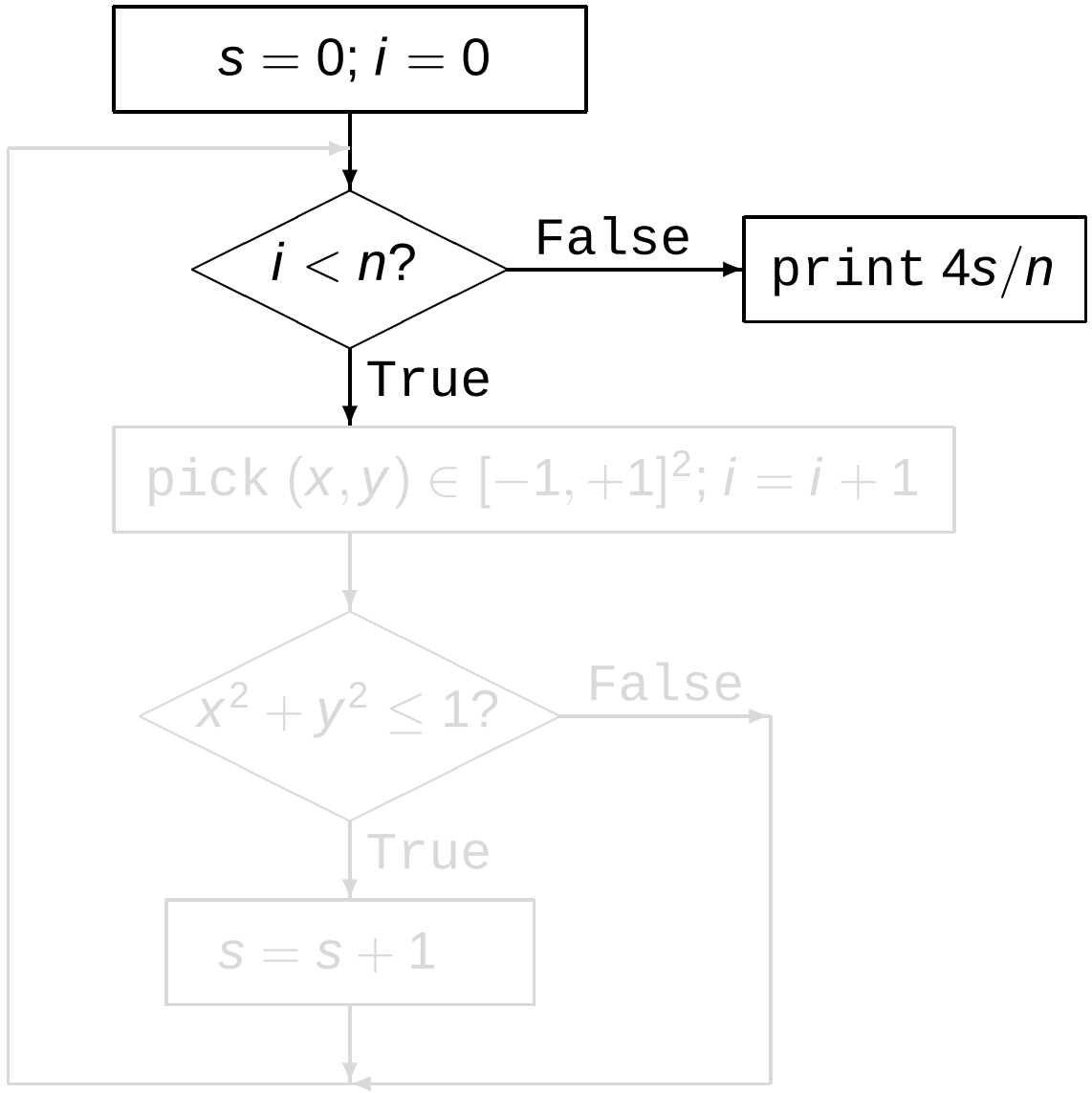
- Expected values are expressed as integrals.
  When many parameters are involved, the integration is high dimensional and only estimation is possible.
- The area of the unit disk is π.



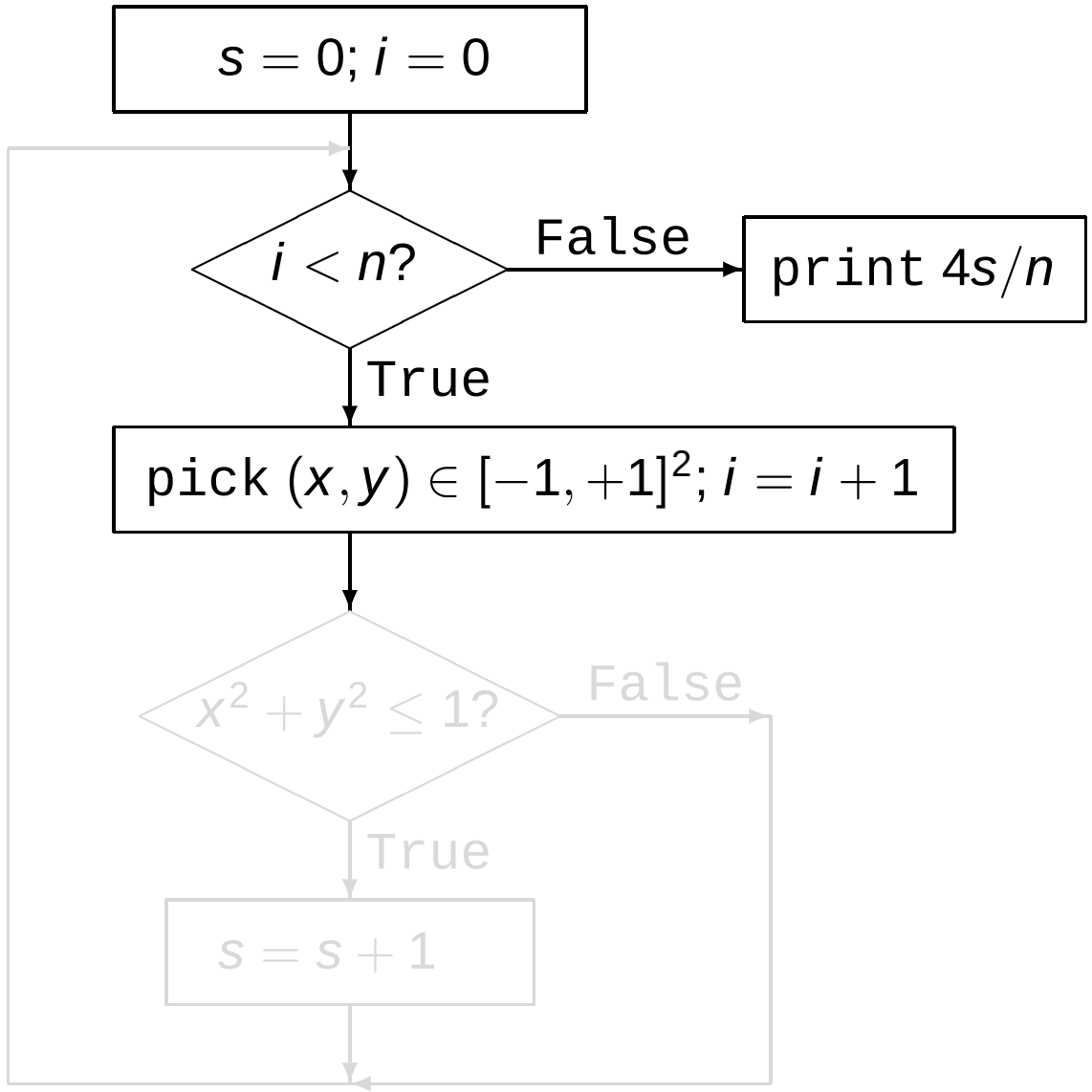
Generate random uniformly distributed points with coordinates $(x, y) \in [-1, +1] \times [-1, +1]$.

We count a success when $x^2 + y^2 \leq 1$.

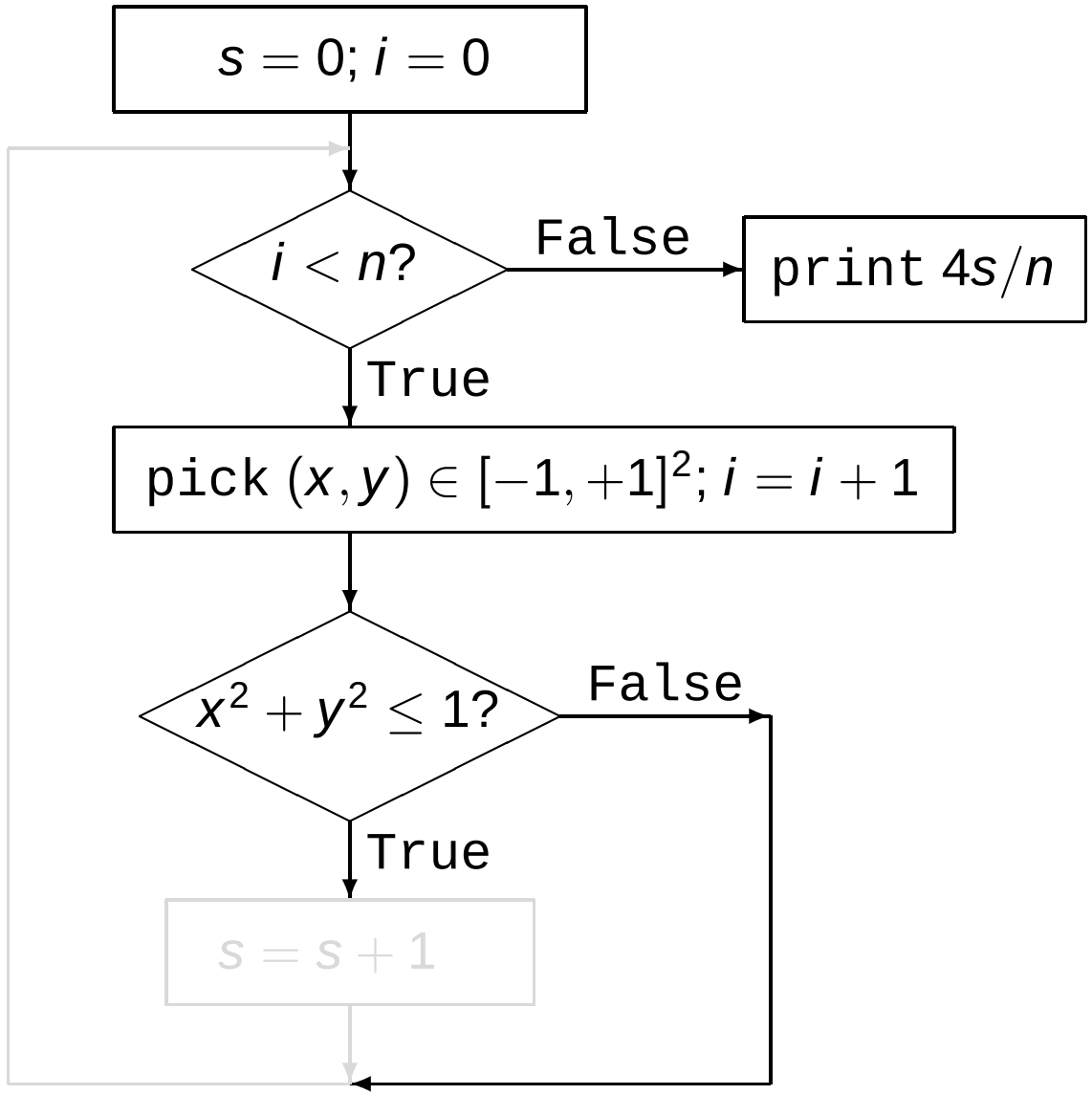
Flowchart for Estimating π



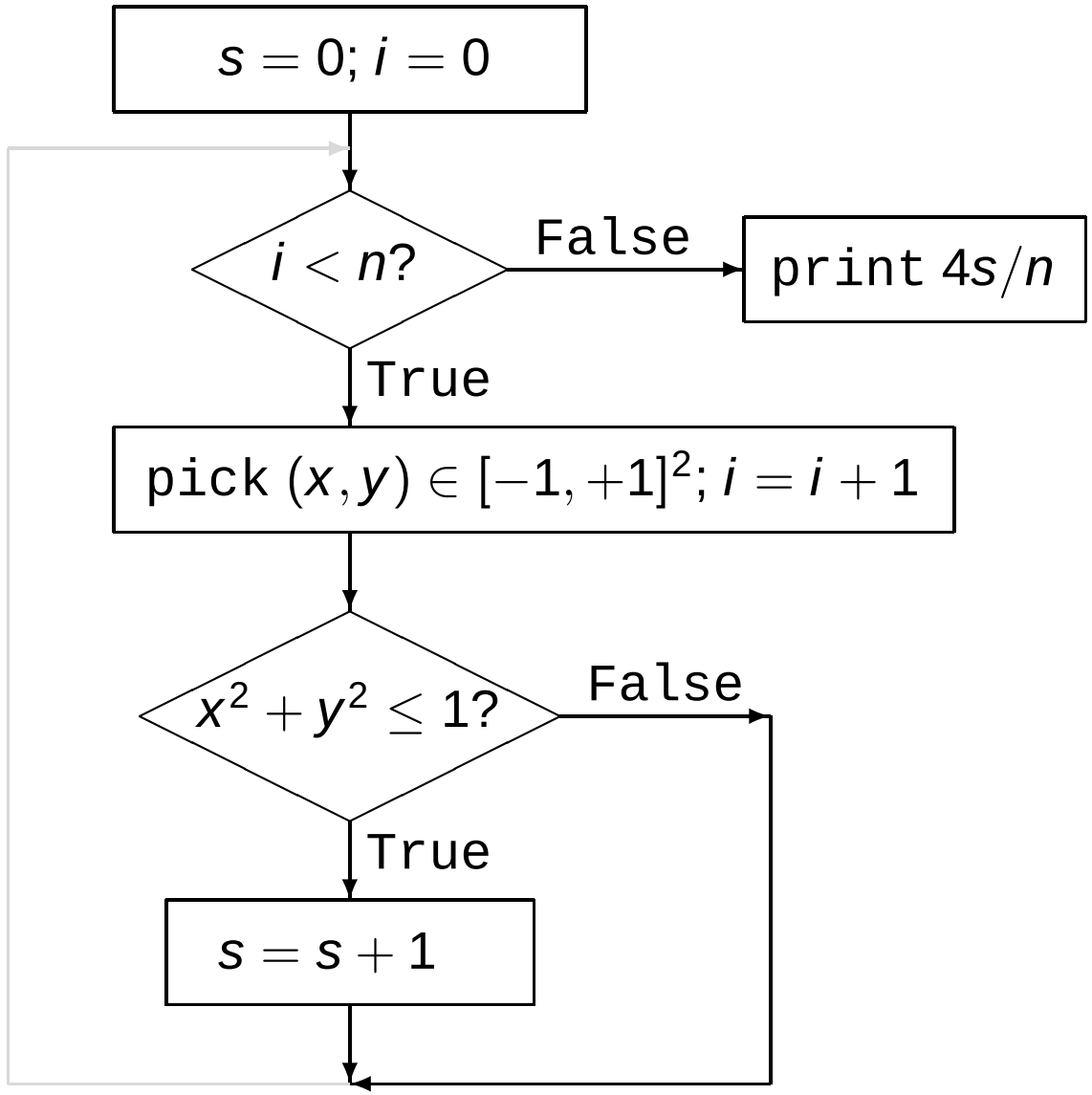
Flowchart for Estimating π



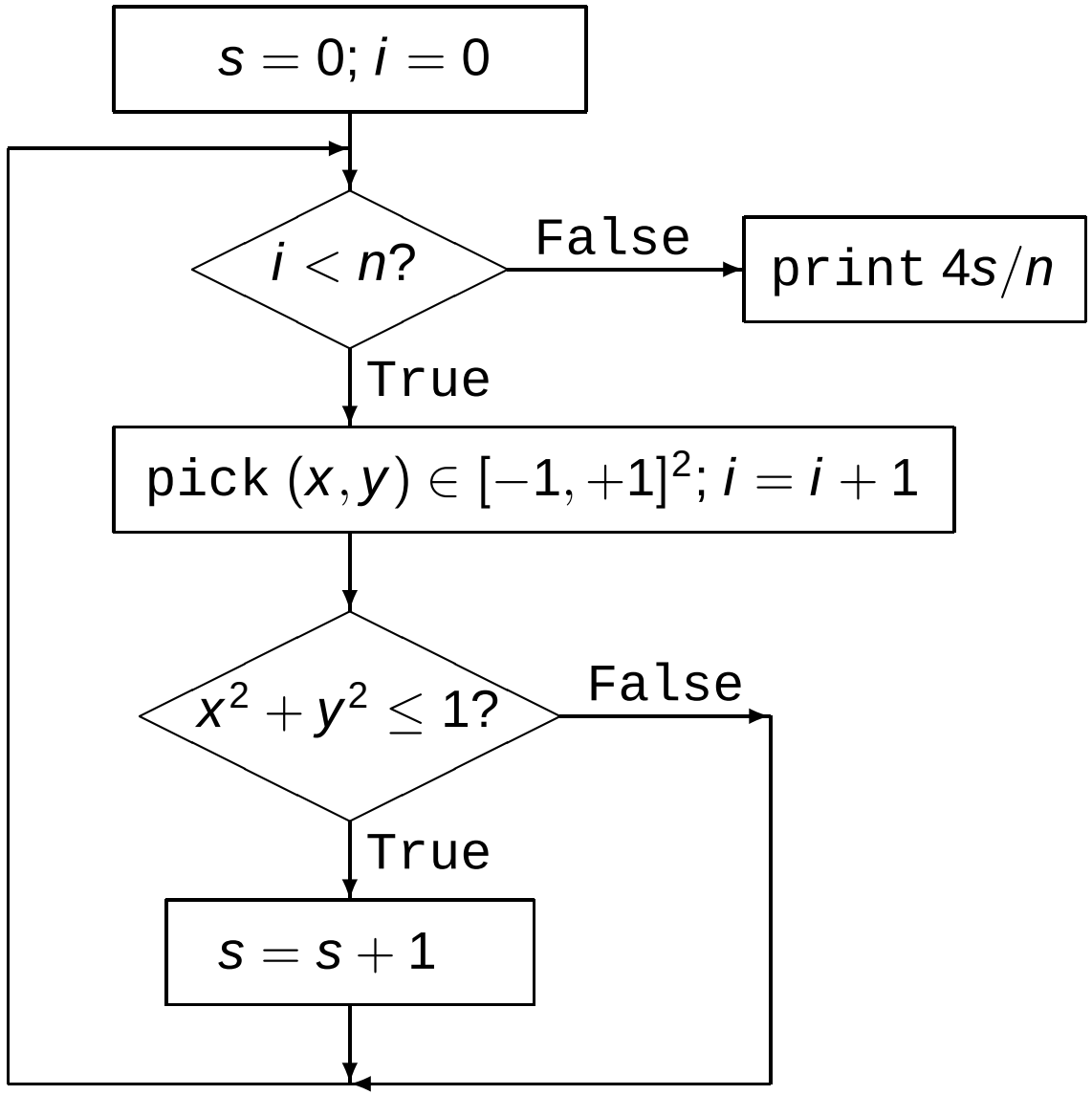
Flowchart for Estimating π



Flowchart for Estimating π



Flowchart for Estimating π



Estimating π

The program `mc4pi.py`:

```
# L-12 MCS 260 Mon 8 Feb 2010  Monte Carlo
#
import random
print 'Monte Carlo simulation for Pi'
n = input('Give number of runs : ')
s = 0
for i in range(0,n):
   x = random.uniform(-1,1)
   y = random.uniform(-1,1)
   if x**2 + y**2 <= 1: s += 1
print 'After %d runs : %f' % (n,4.0*s/n)
```

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

Estimating π

The program `mc4pi.py`:

```
# L-12 MCS 260 Mon 8 Feb 2010  Monte Carlo
#
import random
print 'Monte Carlo simulation for Pi'
n = input('Give number of runs : ')
s = 0
for i in range(0,n):
   x = random.uniform(-1,1)
   y = random.uniform(-1,1)
   if x**2 + y**2 <= 1: s += 1
print 'After %d runs : %f' % (n,4.0*s/n)
```

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

Estimating π

The program `mc4pi.py`:

```
# L-12 MCS 260 Mon 8 Feb 2010  Monte Carlo
#
import random
print 'Monte Carlo simulation for Pi'
n = input('Give number of runs : ')
s = 0
for i in range(0,n):
   x = random.uniform(-1,1)
   y = random.uniform(-1,1)
   if x**2 + y**2 <= 1: s += 1
print 'After %d runs : %f' % (n,4.0*s/n)
```

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

Estimating π

The program mc4pi.py:

```
# L-12 MCS 260 Mon 8 Feb 2010  Monte Carlo
#
import random
print 'Monte Carlo simulation for Pi'
n = input('Give number of runs : ')
s = 0
for i in range(0,n):
   x = random.uniform(-1,1)
   y = random.uniform(-1,1)
   if x**2 + y**2 <= 1: s += 1
print 'After %d runs : %f' % (n,4.0*s/n)
```

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

Running Simulations repeat until: break

Converting Numbers

Converting 123, from decimal into binary format:

n	$n/2$	$n \bmod 2$	
123	61	1	$123 = 61 \times 2 + 1$
61	30	1	$61 = 30 \times 2 + 1$
30	15	0	$30 = 15 \times 2 + 0$
15	7	1	$15 = 7 \times 2 + 1$
7	3	1	$7 = 3 \times 2 + 1$
3	1	1	$3 = 1 \times 2 + 1$
1	0	1	$1 = 0 \times 2 + 1$

$$\begin{aligned} 123 &= 1 + 2 \times 61 = 1 + 2 \times (1 + 2 \times 30) \\ &= 1 + 2 \times (1 + 2 \times (0 + 2 \times 15)) \\ &= 1 + 2 \times (1 + 2 \times (0 + 2 \times (1 + 2 \times 7))) \\ &= \ldots = 1111011 = 7B. \end{aligned}$$

The table shows the progression of the values of the variables in the loop, each row is one iteration.

Binary Expansions

repeat until loops

The bits of a number are the remainders of division by 2.

divmod() is an intrinsic operation:

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

Use as `(n,r) = divmod(n,2)`
to obtain remainder n%2 in `r` and to replace n by n/2.

Pseudo code to compute the binary expansion:

```
n = input()
repeat
   (n,r) = divmod(n,2)
   print r
until (n == 0).
```

Binary Expansions

repeat until loops

The bits of a number are the remainders of division by 2.
divmod() is an intrinsic operation:

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

Use as `(n,r) = divmod(n,2)`
to obtain remainder n%2 in `r` and to replace n by n/2.

Pseudo code to compute the binary expansion:

```
n = input()
repeat
   (n,r) = divmod(n,2)
   print r
until (n == 0).
```

Binary Expansions

repeat until loops

The bits of a number are the remainders of division by 2.
divmod() is an intrinsic operation:

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

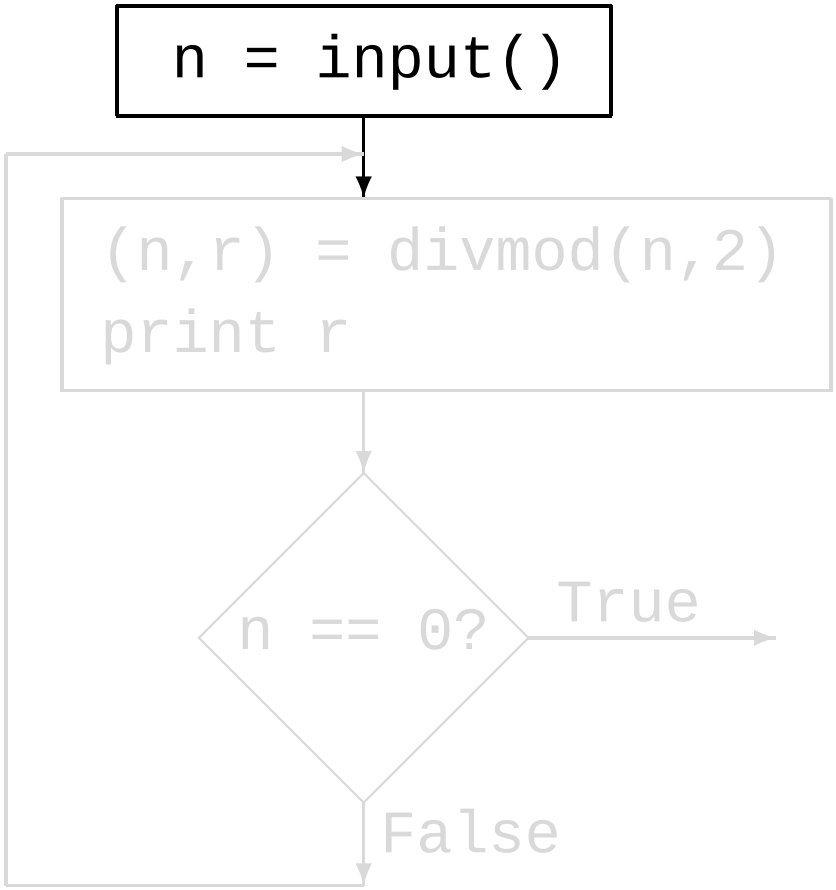
Use as `(n,r) = divmod(n,2)`
to obtain remainder n%2 in `r` and to replace n by n/2.

Pseudo code to compute the binary expansion:

```
n = input()
repeat
   (n,r) = divmod(n,2)
   print r
until (n == 0).
```

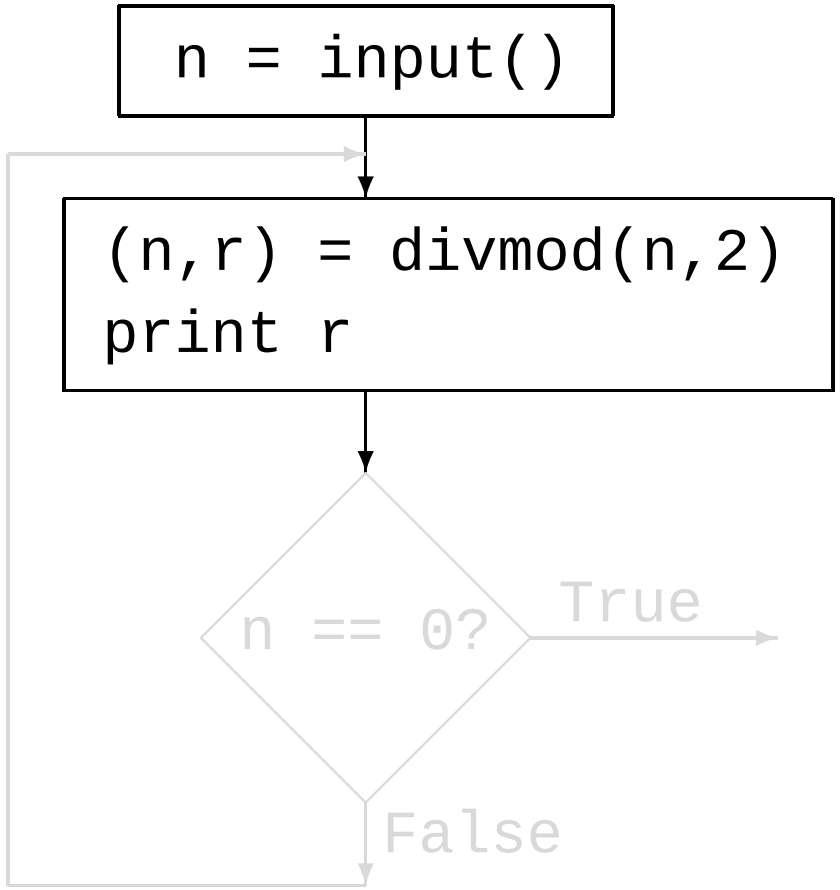
Flowchart of Binary Expansion

picture of repeat until



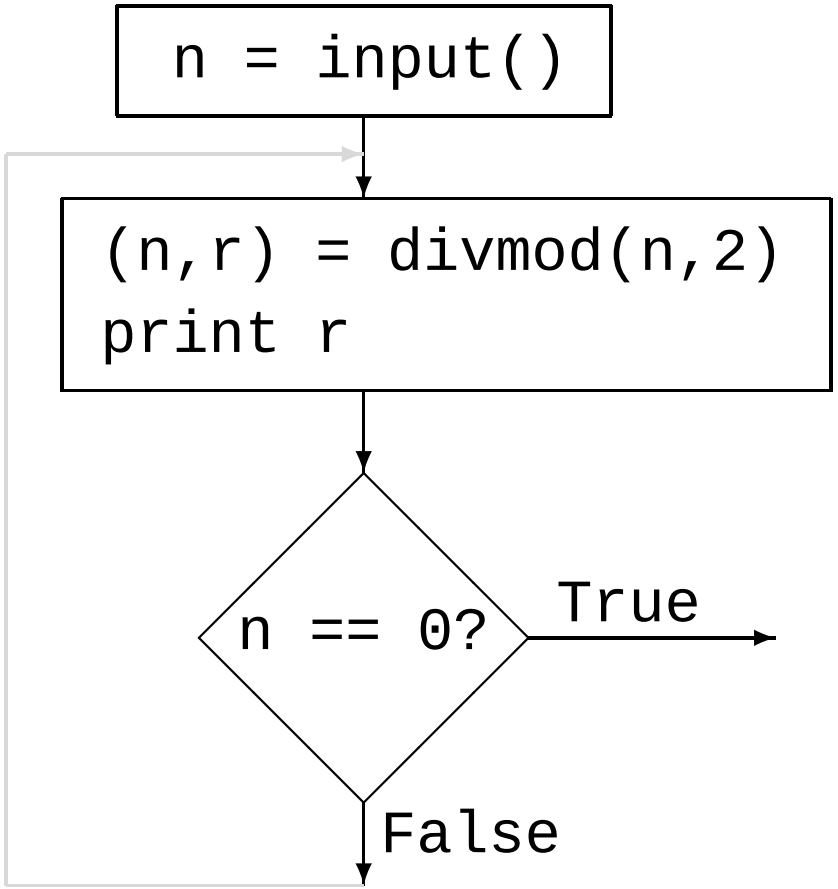
Flowchart of Binary Expansion

picture of repeat until



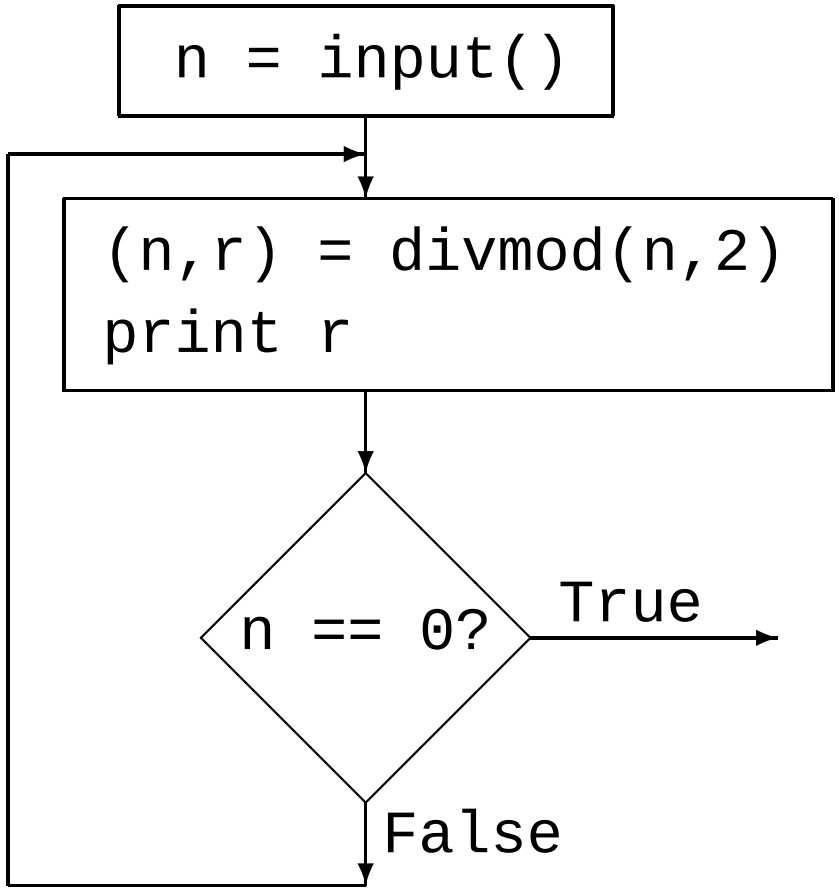
Flowchart of Binary Expansion

picture of repeat until



Flowchart of Binary Expansion

picture of repeat until



A first Python solution

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# This first version prints the bits
# in the order as they are computed.
#
print 'computing the binary expansion'
n = input('Give a number : ')
(n,r) = divmod(n,2)
print r
while n > 0:
   (n,r) = divmod(n,2)
   print r
```

avoid duplication of code

A first Python solution

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# This first version prints the bits
# in the order as they are computed.
#
print 'computing the binary expansion'
n = input('Give a number : ')
(n,r) = divmod(n,2)
print r
while n > 0:
   (n,r) = divmod(n,2)
   print r
```

avoid duplication of code

Running Simulations
repeat until: break

The break Statement

repeat until as while true break

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

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

is realized in Python as

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

The `while True` starts an infinite loop, terminated when `< condition >` becomes True.

The break Statement

repeat until as while true break

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

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

is realized in Python as

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

The `while True` starts an infinite loop, terminated when `< condition >` becomes True.

The break Statement

repeat until as while true break

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

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

is realized in Python as

```
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:

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# Use of break for repeat until.
#
print 'computing the binary expansion'
n = input('Give a number : ')
while True:
    (n,r) = divmod(n,2)
    print r
    if n == 0: break
```

Exercise: how to print bits in correct order?

Binary Expansions with break

a better solution

The program below avoids the duplication of code:

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# Use of break for repeat until.
#
print 'computing the binary expansion'
n = input('Give a number : ')
while True:
   (n,r) = divmod(n,2)
   print r
   if n == 0: break
```

Exercise: how to print bits in correct order?

Binary Expansions with break

a better solution

The program below avoids the duplication of code:

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# Use of break for repeat until.
#
print 'computing the binary expansion'
n = input('Give a number : ')
while True:
   (n,r) = divmod(n,2)
   print r
   if n == 0: break
```

Exercise: how to print bits in correct order?

Binary Expansions with break

a better solution

The program below avoids the duplication of code:

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# Use of break for repeat until.
#
print 'computing the binary expansion'
n = input('Give a number : ')
while True:
   (n,r) = divmod(n,2)
   print r
   if n == 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.

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# A break only effects one loop.
#
print 'computing the binary expansion'
while True:
   n = input('Give a number (< 0 to exit) : ')
   if n < 0: break
   while True:
      (n,r) = divmod(n,2)
      print r
      if n == 0: break
```

A break only effects the one loop it is in.

Two Loops, Two Breaks

scope of a break

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

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# A break only effects one loop.
#
print 'computing the binary expansion'
while True:
   n = input('Give a number (< 0 to exit) : ')
   if n < 0: break
   while True:
      (n,r) = divmod(n,2)
      print r
      if n == 0: break
```

A break only effects the one loop it is in.

Two Loops, Two Breaks

scope of a break

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

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# A break only effects one loop.
#
print 'computing the binary expansion'
while True:
   n = input('Give a number (< 0 to exit) : ')
   if n < 0: break
   while True:
      (n,r) = divmod(n,2)
      print r
      if n == 0: break
```

A break only effects the one loop it is in.

Two Loops, Two Breaks

scope of a break

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

```
# L-12 MCS 260 Mon 8 Feb 2010  binary expansion
#
# A break only effects one loop.
#
print 'computing the binary expansion'
while True:
   n = input('Give a number (< 0 to exit) : ')
   if n < 0: break
   while True:
      (n,r) = divmod(n,2)
      print r
      if n == 0: break
```

A break only effects the one loop it is in.

Running Simulations repeat until: break

Arrays represent Matrices

using numpy

Python has no array type, we use `numpy`:

```
>>> from numpy import *
>>> A = zeros((2,3),int)
>>> A
array([[0, 0, 0],
       [0, 0, 0]])
```

We can define arrays via a list of lists:

```
>>> B = array([[1,2,3],[4,5,6]])
>>> B
array([[1, 2, 3],
       [4, 5, 6]])
```

Two ways of accessing elements:

```
>>> B[1][1]
5
>>> B[1,1]
5
```

Arrays represent Matrices

using numpy

Python has no array type, we use `numpy`:

```
>>> from numpy import *
>>> A = zeros((2,3),int)
>>> A
array([[0, 0, 0],
       [0, 0, 0]])
```

We can define arrays via a list of lists:

```
>>> B = array([[1,2,3],[4,5,6]])
>>> B
array([[1, 2, 3],
       [4, 5, 6]])
```

Two ways of accessing elements:

```
>>> B[1][1]
5
>>> B[1,1]
5
```

Arrays represent Matrices

using numpy

Python has no array type, we use `numpy`:

```
>>> from numpy import *
>>> A = zeros((2,3),int)
>>> A
array([[0, 0, 0],
       [0, 0, 0]])
```

We can define arrays via a list of lists:

```
>>> B = array([[1,2,3],[4,5,6]])
>>> B
array([[1, 2, 3],
       [4, 5, 6]])
```

Two ways of accessing elements:

```
>>> B[1][1]
5
>>> B[1,1]
5
```

Random Arrays

A 2-by-3 array with elements in $[-10, +10]$:

```
>>> X = random.randint(-10,10,(2,3))
>>> X
array([[ -5, -10,   0],
       [  7,   1,  -5]])
```

Random arrays of floats:

```
>>> Y = random.rand(2,3)
>>> Y
array([[ 0.43049771,  0.62879832,  0.54863368],
       [ 0.37764022,  0.32186191,  0.43726834]])
>>> Z = random.randn(2,3)
>>> Z
array([[-0.55872826, -0.342225  ,  0.50209898],
       [ 1.05289782, -0.01733416, -0.29531646]])
```

Random Arrays

A 2-by-3 array with elements in $[-10, +10]$:

```
>>> X = random.randint(-10,10,(2,3))
>>> X
array([[ -5, -10,   0],
       [  7,   1,  -5]])
```

Random arrays of floats:

```
>>> Y = random.rand(2,3)
>>> Y
array([[ 0.43049771,  0.62879832,  0.54863368],
       [ 0.37764022,  0.32186191,  0.43726834]])
>>> Z = random.randn(2,3)
>>> Z
array([[-0.55872826, -0.342225  ,  0.50209898],
       [ 1.05289782, -0.01733416, -0.29531646]])
```

Running Simulations
repeat until: break

Search a 2-dimensional Array

Problem statement:

Input: A is an array 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
2. The computer generates n-by-m array A of random integer numbers in the interval $[-100, +100]$.
3. The program prompts the user for x, and
4. searches A for x and prints search result.

Search a 2-dimensional Array

Problem statement:

Input: A is an array 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
2. The computer generates n-by-m array A of random integer numbers in the interval $[-100, +100]$.
3. The program prompts the user for x, and
4. searches A for x and prints search result.

Search a 2-dimensional Array

Problem statement:

Input: A is an array 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
2. The computer generates n-by-m array A of random integer numbers in the interval $[-100,+100]$.
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 number of rows : 4
give number of columns : 5
a random 4-by-5 array :
[[-13  -9 -35 -82 -21]
 [-46  99  62  93  37]
 [ 88  47  66 -12  36]
 [-59  95  53 -97 -93]]

give a number : 47
found 47 at [2,1]
```

If the given number does not occur, then
`'%d does not occur in array'` is printed.

Running the Code

at the command prompt

```
$ python findelem.py
give number of rows : 4
give number of columns : 5
a random 4-by-5 array :
[[-13  -9 -35 -82 -21]
 [-46  99  62  93  37]
 [ 88  47  66 -12  36]
 [-59  95  53 -97 -93]]

give a number : 47
found 47 at [2,1]
```

If the given number does not occur, then
`'%d does not occur in array'` is printed.

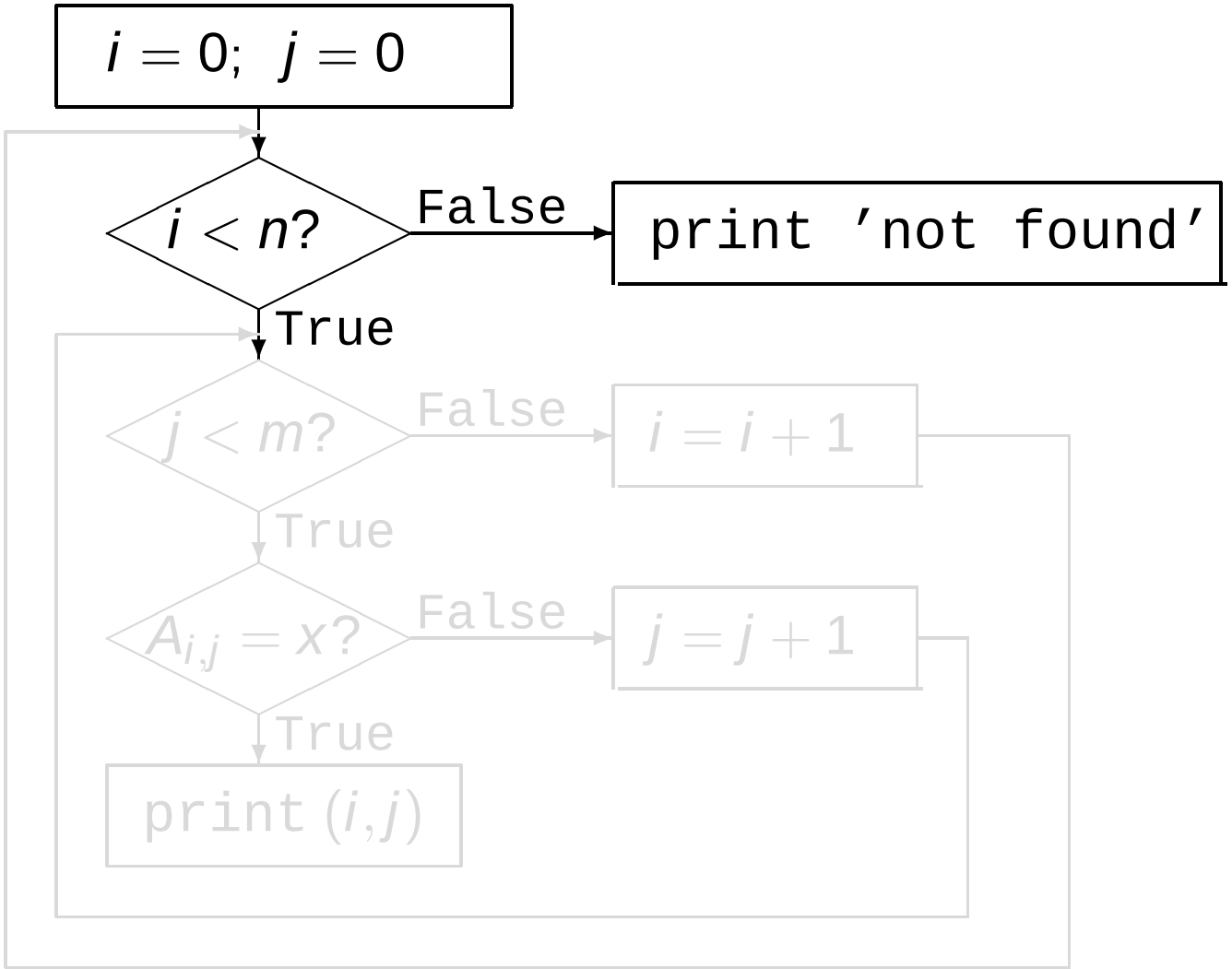
Running the Code

at the command prompt

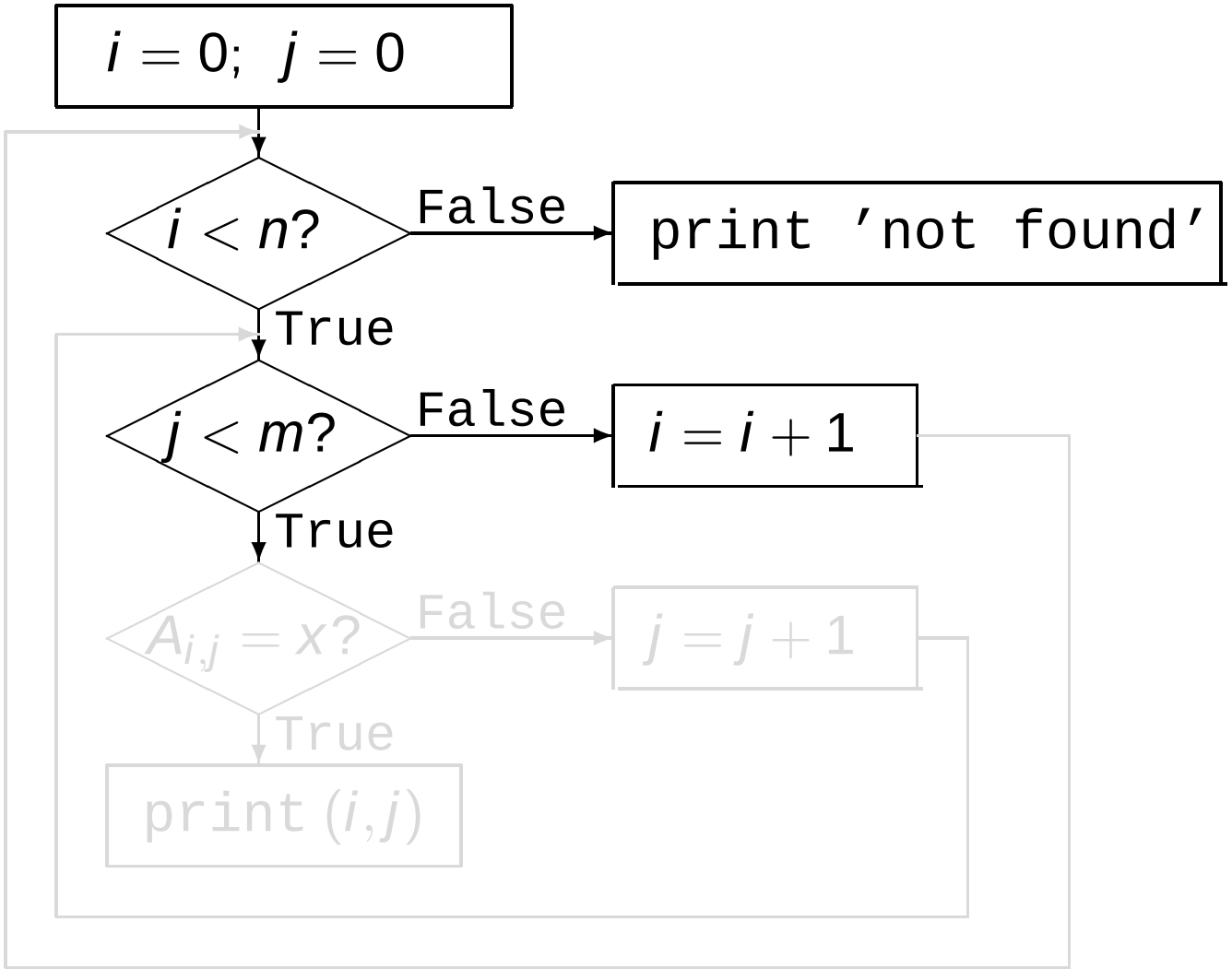
```
$ python findelem.py
give number of rows : 4
give number of columns : 5
a random 4-by-5 array :
[[-13  -9 -35 -82 -21]
 [-46  99  62  93  37]
 [ 88  47  66 -12  36]
 [-59  95  53 -97 -93]]

give a number : 47
found 47 at [2,1]
```

If the given number does not occur, then
`'%d does not occur in array'` is printed.

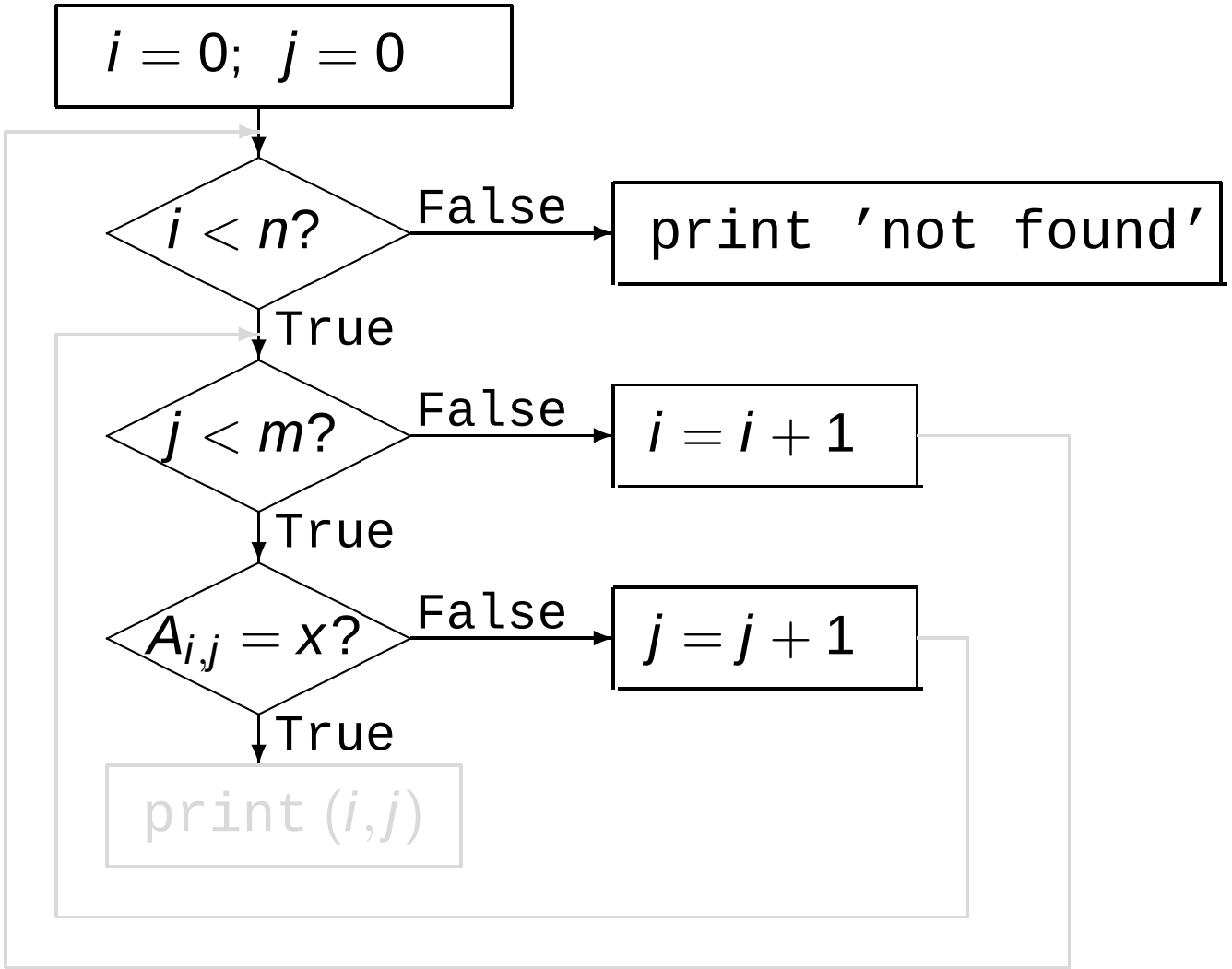
Flowchart



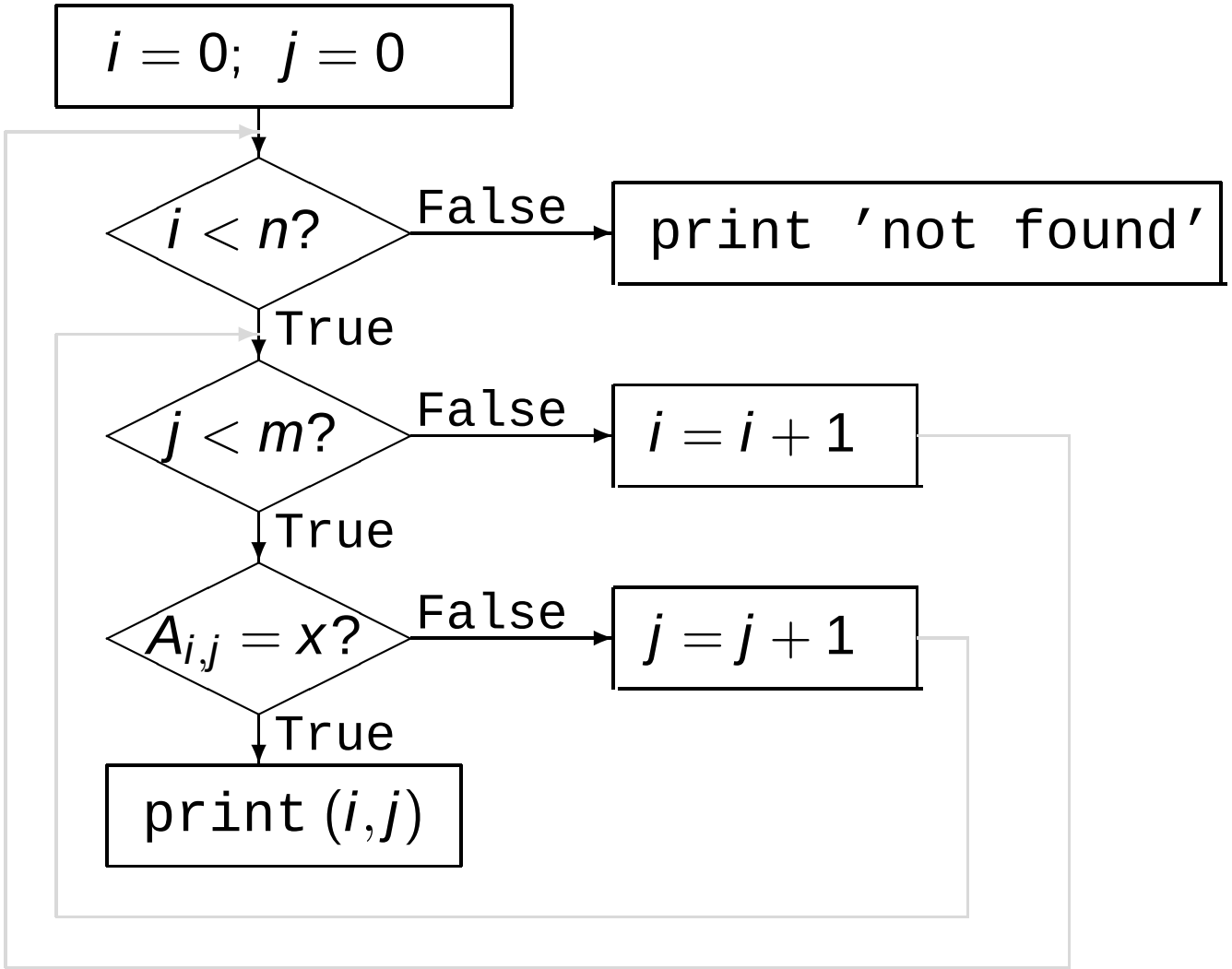
Flowchart



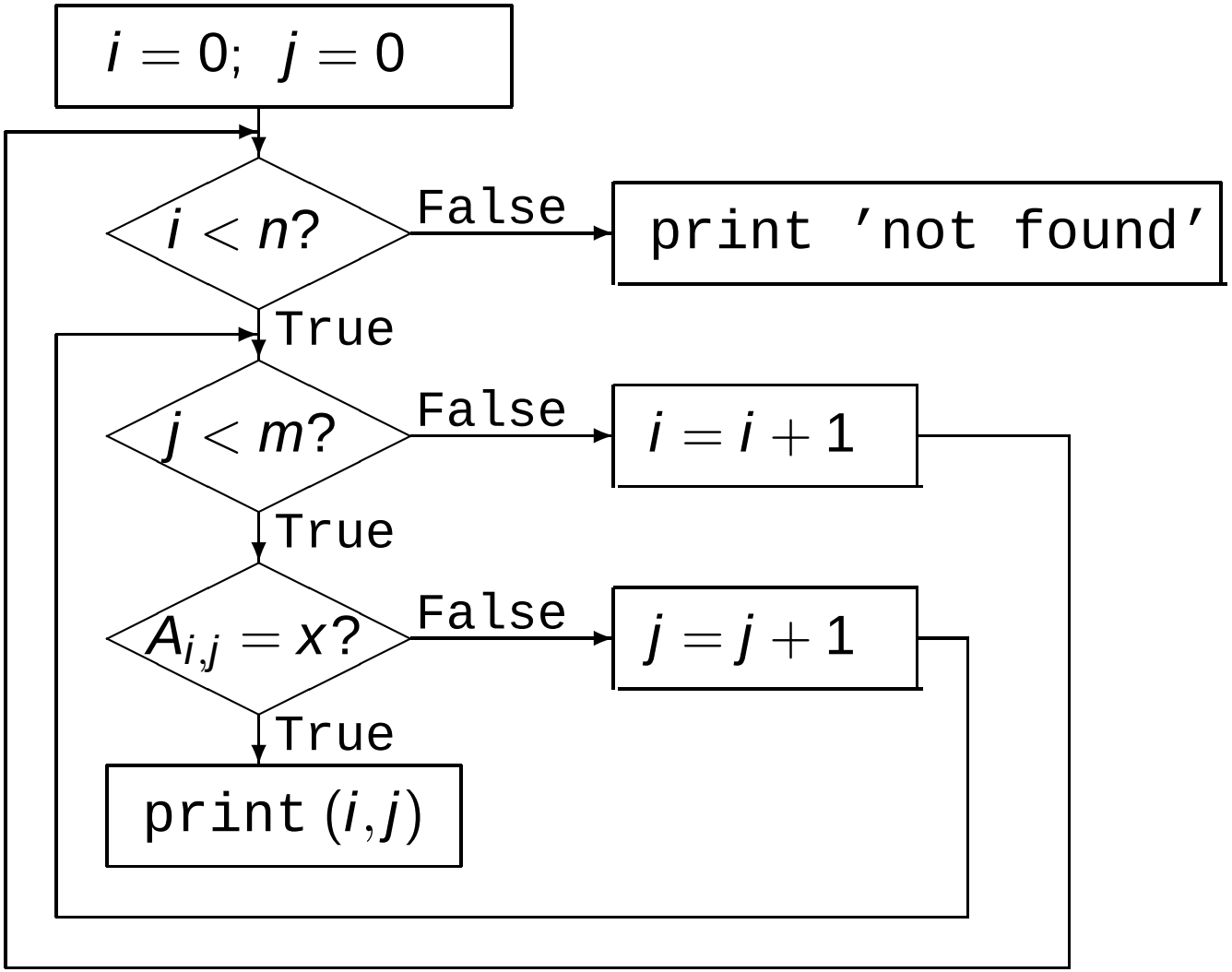
Flowchart



Flowchart



Flowchart



Start of findelem.py

```
# L-12 MCS 260 Mon 8 Feb 2010 : find element

# Illustration of a double for loop to find
# an element in a two dimensional array.

from numpy import *
n = input('give number of rows : ')
m = input('give number of columns : ')
A = random.randint(-100,100,(n,m))
print 'a random %d-by-%d array :\n' % (n,m), A
x = input('give a number : ')
```

Start of findelem.py

```
# L-12 MCS 260 Mon 8 Feb 2010 : find element

# Illustration of a double for loop to find
# an element in a two dimensional array.

from numpy import *
n = input('give number of rows : ')
m = input('give number of columns : ')
A = random.randint(-100,100,(n,m))
print 'a random %d-by-%d array :\n' % (n,m), A
x = input('give a number : ')
```

the double for loop

Search *n*-by-*m* array *A* for *x*:

```
found = False
for i in range(0,n):
   for j in range(0,m):
      if A[i,j] == x:
         found = True
         break
   if found: break
if found:
   print 'found %d at [%d,%d]' % (x,i,j)
else:
   print '%d does not occur in array' % x
```

the double for loop

Search *n*-by-*m* array *A* for *x*:

```
found = False
for i in range(0,n):
   for j in range(0,m):
      if A[i,j] == x:
         found = True
         break
   if found: break
if found:
   print 'found %d at [%d,%d]' % (x,i,j)
else:
   print '%d does not occur in array' % x
```

the double for loop

Search *n*-by-*m* array *A* for *x*:

```
found = False
for i in range(0,n):
   for j in range(0,m):
      if A[i,j] == x:
         found = True
         break
   if found: break
if found:
   print 'found %d at [%d,%d]' % (x,i,j)
else:
   print '%d does not occur in array' % x
```

Assignments

1. Use a stack to store the bits in the binary expansion to print the bits *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:
   ≥ 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 π. 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*.