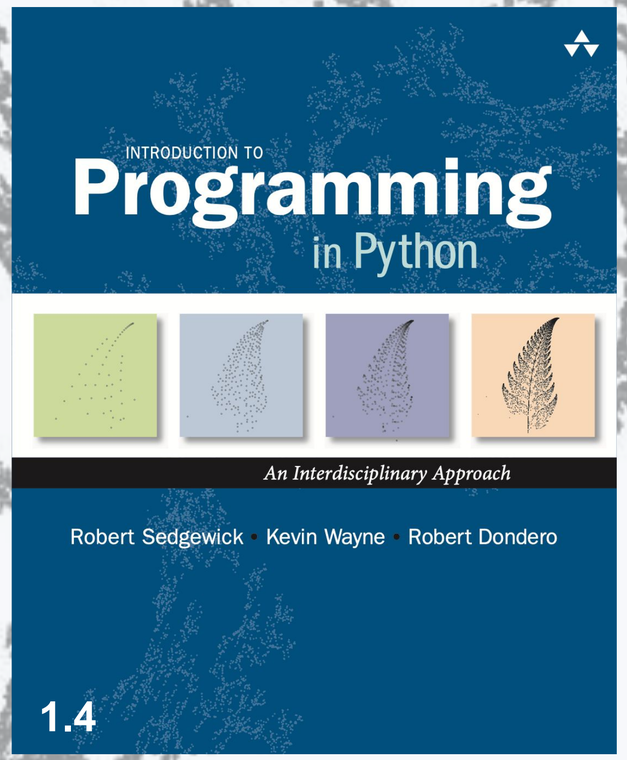
INTRO TO PROGRAMMING IN PYTHON

SEDGEWICK · WAYNE · DONDERO



https://introcs.cs.princeton.edu/python

3. Arrays

INTRO TO PROGRAMMING IN PYTHON

SEDGEWICK • WAYNE • DONDERO

3. Arrays

- Basic concepts
- Typical array-processing code
- Two-dimensional arrays

A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math | text I/O

primitive data types | assignment statements

Ability to store and process huge amounts of data

Arrays

A data structure is an arrangement of data that enables efficient processing by a program.

An array is an *indexed* sequence of values typically of the same type.

Examples.

- 52 playing cards in a deck.
- 100 thousand students in an online class.
- 1 billion pixels in a digital image.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 86 billion neurons in the brain.
- 50 trillion cells in the human body.

index	*value*
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠

Note that indexing starts at 0 and ends at length - 1

Main purpose. Facilitate storage and manipulation of data.

Processing many values of the same type

10 values, without arrays

```
a0 = 0.0
a1 = 0.0
a2 = 0.0
a3 = 0.0
a4 = 0.0
a5 = 0.0
a6 = 0.0
a7 = 0.0
a8 = 0.0
a9 = 0.0
...
a4 = 3.0
...
a8 = 8.0
...
x = a4 + a8
```

tedious and error-prone code

10 values, with an array

```
a = stdarray.create1D(10, 0.0)
...
a[4] = 3.0
...
a[8] = 8.0
...
x = a[4] + a[8]
```

an easy alternative

1 million values, with an array

```
stdarray.create1D(1000000, 0.0)
...
a[234567] = 3.0
...
a[876543] = 8.0
...
x = a[234567] + a[876543]
```

scales to handle huge amounts of data

Python Lists vs Arrays and the stdarray module

Arrays are fixed-length data structures. Python does not have an array data structure. A Python *list* is similar (creation, indexed access, indexed assignment, iteration) to an array but variable-length.

Python list (variable-length)

list

```
x = []          # len(x) = 0
x += [0.30]     # len(x) = 1
x += [0.60]     # len(x) = 2
x += [0.10]     # len(x) = 3
x += [0.99]     # len(x) = 4
```

list concatenation (iterable)

Java array (fixed-length)

array *initializes to 0.0*

```
double[] x = new double[3]; # x.length = 3
x[0] = 0.30;                # x.length = 3
x[1] = 0.60;                # x.length = 3
x[2] = 0.10;                # x.length = 3
x[3] = 0.99;                # IndexError!
```

The *stdarray* module provides utility methods to create an array of n elements and initialize to a given value. Self-documenting.

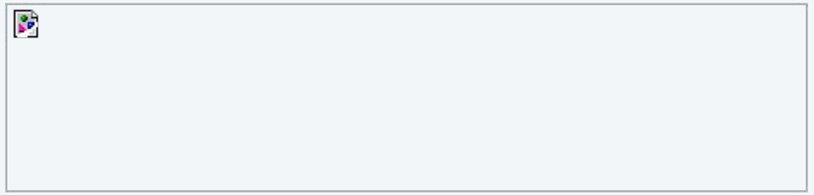
Python "array"

```
x = stdarray.create1D( 3, 0.0 ) # len(x) = 3
x[0] = 0.30                     # len(x) = 3
x[1] = 0.60                     # len(x) = 3
x[2] = 0.10                     # len(x) = 3
x[3] = 0.99                     # IndexError!
```

More Usecases

- Save *precomputed* values.
- Simplify repetitive code.

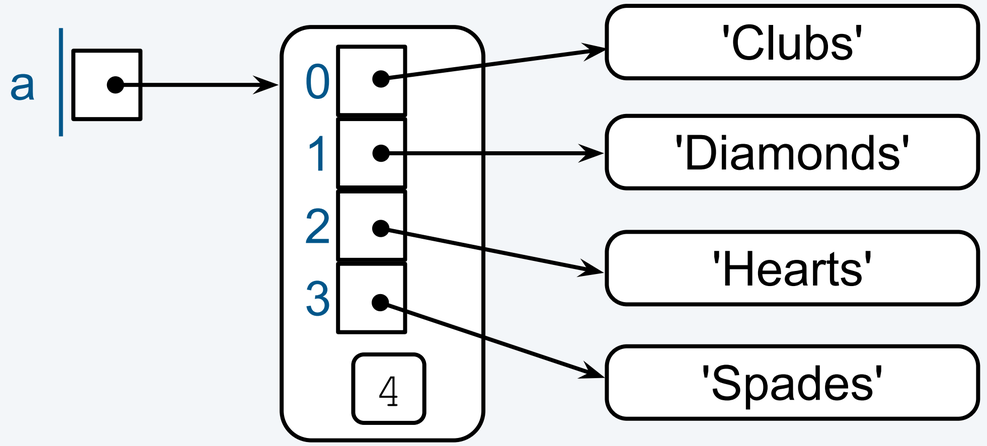
```
m = 1
if   (m == 1): stdio.writeln("Jan")
elif (m == 2): stdio.writeln("Feb")
elif (m == 2): stdio.writeln("Mar")
elif (m == 2): stdio.writeln("Apr")
elif (m == 2): stdio.writeln("May")
elif (m == 2): stdio.writeln("Jun")
elif (m == 2): stdio.writeln("Jul")
elif (m == 2): stdio.writeln("Aug")
elif (m == 2): stdio.writeln("Sep")
elif (m == 2): stdio.writeln("Oct")
elif (m == 2): stdio.writeln("Nov")
elif (m == 2): stdio.writeln("Dec")
```



it does not copy the array.

Memory representation of an array

Recall that variables point to objects in memory. An array is a data structure that stores variables (implicit name) contiguously as a sequence of pointers. This arrangement corresponds to the computer's memory locations (see next slide).



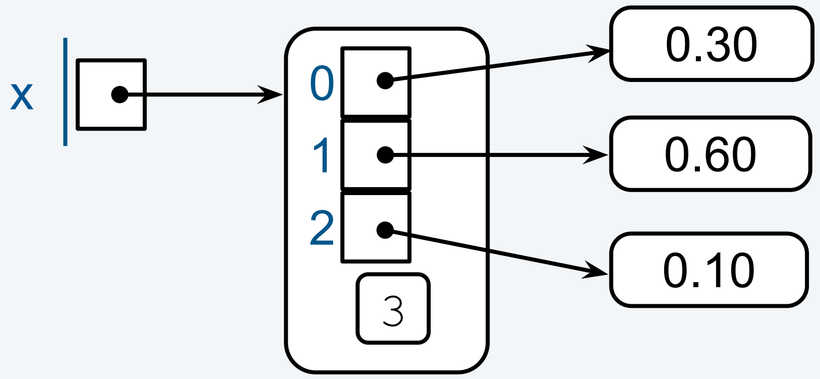
Critical concepts

- Indices start at 0.
- Given `i`, the operation of accessing the value `a[i]` is extremely efficient.
- The assignment `b = a` makes the names `b` and `a` refer to the same array.

it does not copy the array

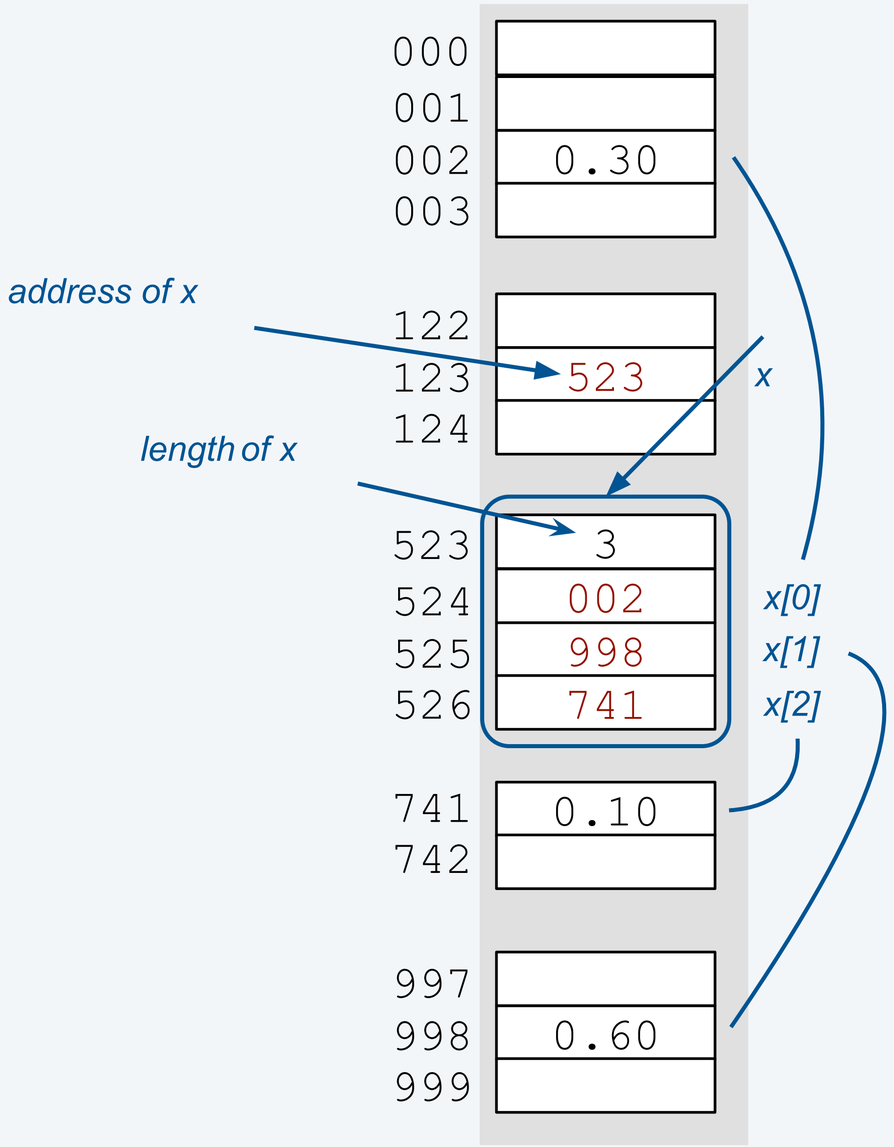
Memory representation of an array

Three useful array memory representations.



1. Object Viewpoint

3. Simplified Memory Viewpoint
Least exact but compact



2. Computer Memory Viewpoint

Creating and initializing Arrays

Basic support

operation	*typical code*
Create and initialize an array of a given length	a = stdarray.create1D(1000,0.0)
Assign and access an array entry by index	a[i] = b[j] + c[k]
Refer to the length of an array	`len(`a)

Declare, create and initialize options

operation	*typical code*
Initialize to float type	a = stdarray.create1D(1000,0.0)
Initialize to str literal values	`letters` = ['a', 'b', 'c']
Initialize to int literal values	`ids` = [143, 847, 91093]

Python list equivalent

a = []

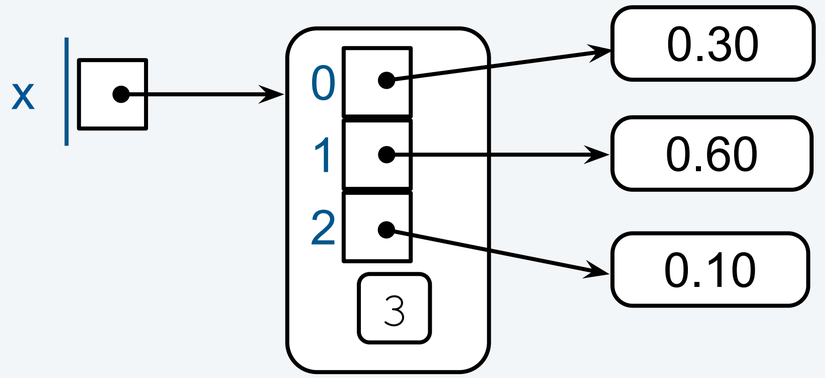
```
for i in range(1000):
   a += [0.0]
```

BUT cost of creating an array is proportional to its length.

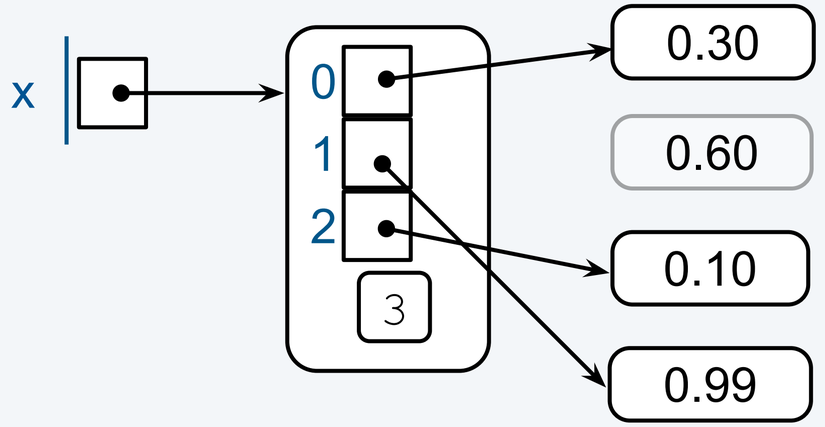
Array Mutability

Reassigning an array element.

```
x = [ 0.30, 0.60, 0.10 ]
...
```




```
...
x[1] = 0.99
...
```



Refer to text for reversing an array.

Array Iteration

Iterate (*visit*) each element in array array element.

Computes average of values in x

```
x = [ 0.30, 0.60, 0.10 ]
total = 0.0
for i in range( len(x) ):
    value = x[i]
    total += value
average = total / len(x)
```

Without explicit indices

```
x = [ 0.30, 0.60, 0.10 ]
total = 0.0
for value in x:
    total += value
average = total / len(x)
```

More compact if only sequence is important but not the index.

Can write elements in array to standard output as one line.

```
x = [ 0.30, 0.60, 0.10 ]
stdio.writeln(x)
```

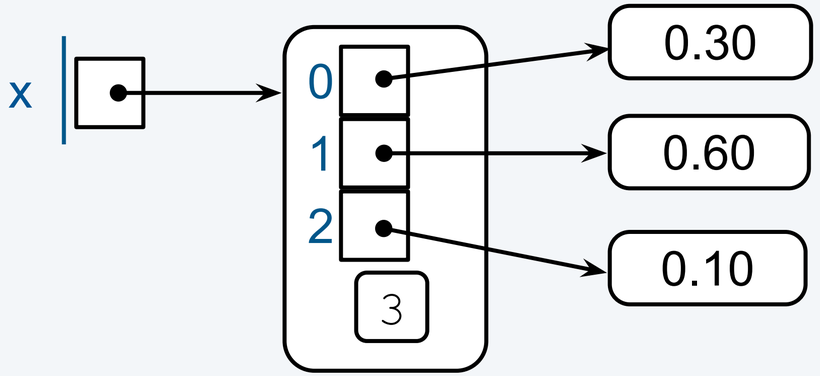
```
>>> [0.3, 0.6, 0.1]
```

No control over format. Iterate to provide explicit formatting

Array Aliases

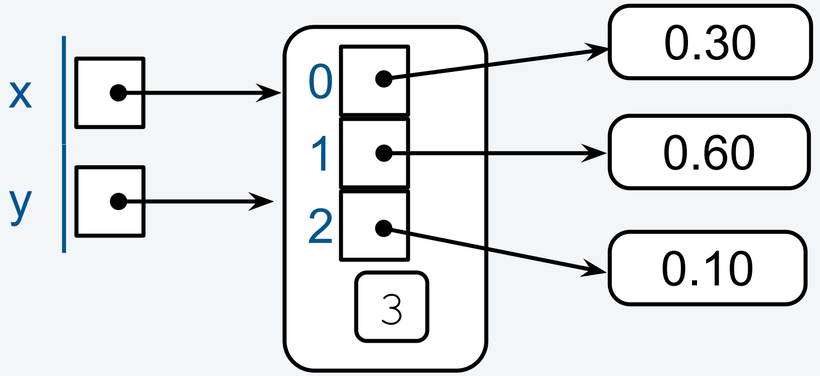
Aliasing when two variables refer to the same object.

```
x = [ 0.30, 0.60, 0.10 ]
```



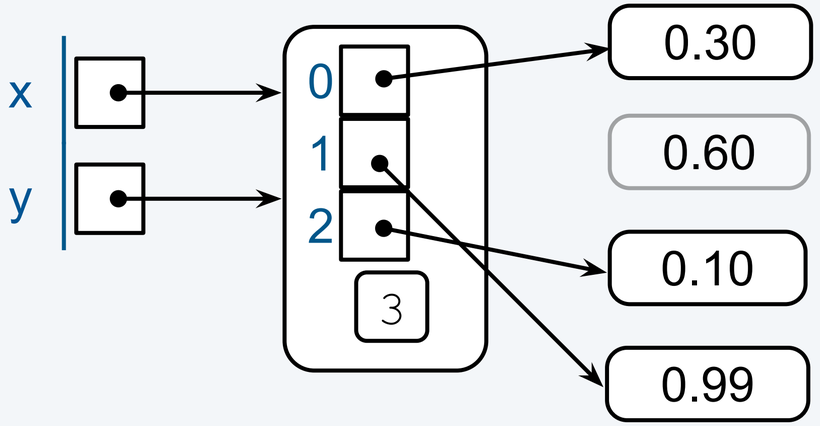
Aliasing mutable objects (arrays are mutable) is confusing and should generally be avoided.

```
y = x
```



Note that str, int, float, and bool are immutable and OK to have multiple variables refer to them.

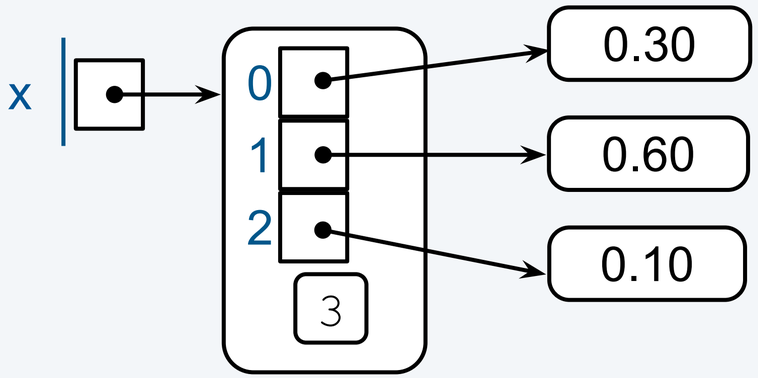
```
x[1] = 0.99
```



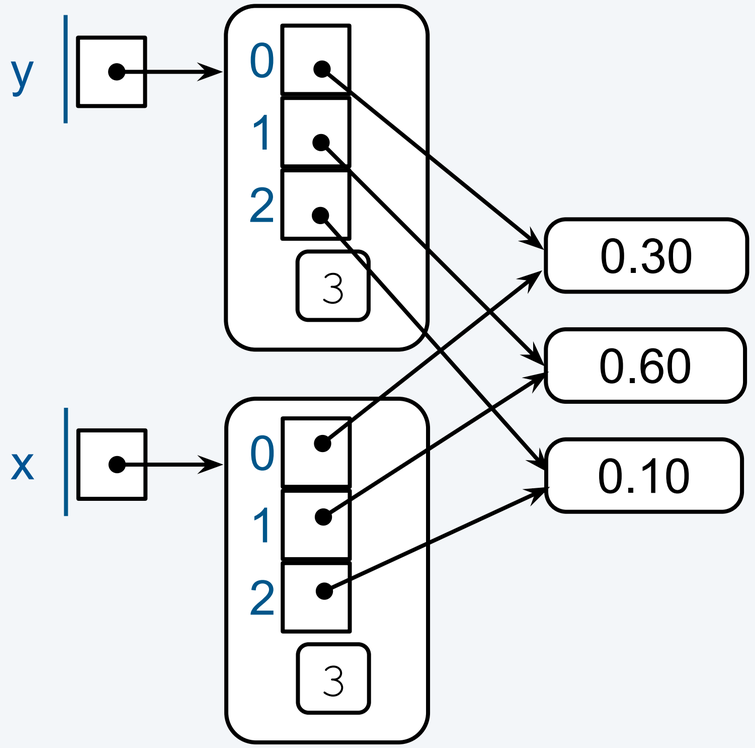
What is y[1]?

Array Copying

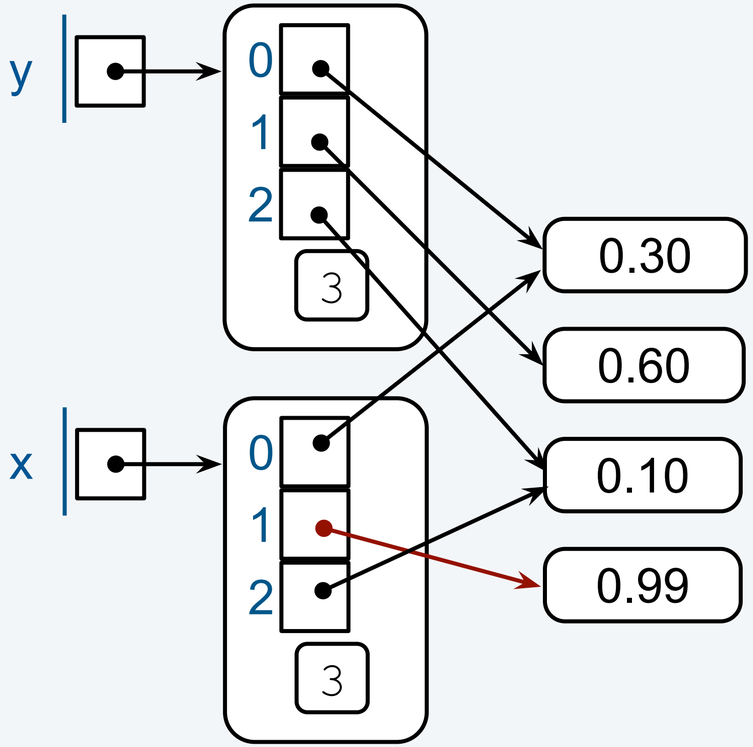
```
x = [ 0.30, 0.60, 0.10 ]
```




```
y = []
for v in x:
    y += [v]
```




```
x[1] = 0.99
```



Array Slicing

Slicing is shorthand (*syntactic sugar*) for creating a new array from an existing array.

```
x = [ 0.30, 0.60, 0.10 ]
i = 0
j = len(x)
y = x[i:j] # [i,j)
```

```
x = [ 0.30, 0.60, 0.10 ]
i = 0
j = len(x)
# Slicing equivalent
y = []
while( i < j ):
    y += [ x[i] ]
    i += 1
```

Slicing default for i is 0 and the default for j is len(x), so can be shortened

```
x = [ 0.30, 0.60, 0.10 ]
y = x[:]
```

```
      0  1  2  3  4  5  6  7
x = [ 5, 2, 8, 6, 5, 6, 2, 4 ]
y = x[2:5]
```

Pop quiz what is y?

```
y = [ 8, 6, 5 ]
```

Though less typing same work proportional to len(x) for the computer.

Programming with arrays: typical examples

For brevity, `N` is desired length for array.

Access command-line args in system array

```
exe    =     sys.argv[0]
stake  = int(sys.argv[1])
goal   = int(sys.argv[2])
trials = int(sys.argv[3])
```

Copy to another array

```
b = stdarray.create1D(N,0.0)
for i in range(N):
   b[i] = a[i]
```

Create an array with N random values

```
a = stdarray.create1D(N,0.0)
for i in range(N):
   a[i] = stdrandom.uniform()
```

Print array values, one per line

```
for i in range(N):
   stdio.writeln(a[i])
```

Compute the average of array values

```
sum = 0.0
for i in range(N):
   sum += a[i]
average = sum / N
```

Find the maximum of array values

```
max = a[0]
for i in range(1,N):
   if (a[i] > max): max = a[i]
```

Vector Dot Product

Dot product. Given two vectors **x[]** and **y[]** of length **N**, their dot product is the sum of the products of their corresponding components.

```
x = [ 0.3, 0.6, 0.1 ]
y = [ 0.5, 0.1, 0.4 ]
N = len(x)
sum = 0.0
for i in range(0,N):
    sum = sum + x[i]*y[i]
```

i	x[i]	y[i]	x[i]*y[i]	sum
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

Programming with Arrays: Typical Bugs

IndexError

```
a = stdarray.create1D( 10, 0.0 )
for i in range( 1, 11 ):
   a[i] = stdrandom.uniform()
```

No a[10] (and a[0] unused)

Uninitialized array
IndexError

```
a = []
for i in range( 0, 10 ):
   a[i] = stdrandom.uniform()
```

Never created the array

Invalid list concatenation
TypeError: 'float' object is not iterable

```
a = []
for i in range( 0, 10 ):
   a[i] += stdrandom.uniform()
```

Can you concatenate a list + float?

Summary

operation	*operator*	*description*
indexed access	`a[i]`	*ith element in* `a[]`
indexed assignment	`a[i] = x`	*replace ith element in* `a[]` *with* `x`
iteration	`for v in a:`	*assign* `v` *to each of the elements in* `a[]`
slicing	`a[i:j]`	*a new array* `[a[i], a[i+1], ..., a[j-1]]` *(*`i` *defaults to 0 and* `j` *defaults to* `len(a)`*)*

operation	*function call*	*description*
length	`len(a)`	*number of elements in* `a[]`
sum	`sum(a)`	*sum of elements in* `a[]`
minimum	`min(a)`	*a minimum element in* `a[]`
maximum	`max(a)`	*a maximum element in* `a[]`

Note: Array elements must be numeric for `sum()` *and comparable for* `min()` *and* `max()`.

Array operations and built-in functions

INTRO TO PROGRAMMING IN PYTHON

SEDGEWICK • WAYNE • DONDERO

3. Arrays

- Basic concepts
- Examples of array-processing code
- Two-dimensional arrays

Shuffling a Deck

Setting Array Values at Compile Time

Ex. Print a random card.

```
RANK = ["2", "3", "4", "5", "6", "7", "8", "9",
        "10", "Jack", "Queen", "King", "Ace"]
SUIT = ["Clubs", "Diamonds", "Hearts", "Spades"]

i = random.randrange(0,len(RANK))     // between 0 and 12
j = random.randrange(0,len(SUIT))    // between 0 and 3

stdio.writeln(RANK[i] + " of " + SUIT[j])
```

Exercise: Printing Deck

Ex. Create a deck of playing cards and print them out.

```
deck = []
for rank in RANK:
    for suit in SUIT:
        card = rank  + " of " + suit
        deck += [card]

for i in range(0, len(RANK)* len(SUIT)):
   stdio.writeln(deck[i])
```

typical array-processing code changes values at runtime

Q. In what order does it output them?

A.

```
two of clubs
two of diamonds
two of hearts
two of spades
three of clubs
...
```

B.

```
two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
...
```

Exercise: Printing Deck

Ex. Create a deck of playing cards and print them out.

```
deck = []
for rank in RANK:
    for suit in SUIT:
        card = rank  + " of " + suit
        deck += [card]

for i in range(0, len(RANK)* len(SUIT)):
    stdio.writeln(deck[i])
```

typical array-processing code changes values at runtime

Q. In what order does it output them?

A.

```
two of clubs
two of diamonds
two of hearts
two of spades
three of clubs
...
```

B.

```
two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
...
```

Shuffling

Goal. Given an array, rearrange its elements in random order.

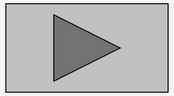
Shuffling algorithm.

- In iteration `i`, pick random card from `deck[i]` through `deck[N-1]`, with each card equally likely.
- Exchange it with `deck[i]`.

```
N = len(deck)
for i in range(0, N):
    r = random.randrange(i,N)
    t = deck[r]
    deck[r] = deck[i]
    deck[i] = t
```

between i and N-1

swap idiom



Shuffling a Deck of Cards: deckshuffle.py

```
import sys
import stdio
import random

def main():
    RANK = ["2", "3", "4", "5", "6", "7", "8", "9",
            "10", "Jack", "Queen", "King", "Ace"]
    SUIT = ["Clubs", "Diamonds", "Hearts", "Spades"]

    deck = []
    for rank in RANK:
        for suit in SUIT:
            card = rank  + " of " + suit
            deck += [card]

    N = len(deck)
    for i in range(0, N):
        r = random.randrange(i,N)
        t = deck[r]
        deck[r] = deck[i]
        deck[i] = t

    for i in range(0, N):
        stdio.writeln(deck[i])
if __name__ == "__main__": main()
```

build the deck

shuffle

print shuffled deck

Shuffling a Deck of Cards

```
% python3 deckshuffling.py
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

```
% python3 deckshuffling.py
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

Without Replacement

Program 1.4.1 Sampling without replacement (sample.py)

```
import random
import sys
import stdarray
import stdio

m = int(sys.argv[1])   # Choose this many elements
n = int(sys.argv[2])   # from 0, 1, ..., n-1.

# Initialize array perm = [0, 1, ..., n-1].
perm = stdarray.create1D(n, 0)
for i in range(n):
    perm[i] = i

# Create a random sample of size m in perm[0..m).
for i in range(m):
    r = random.randrange(i, n)

    # Exchange perm[i] and perm[r].
    temp    = perm[r]
    perm[r] = perm[i]
    perm[i] = temp

# Write the results.
for i in range(m):
    stdio.write(str(perm[i]) + ' ')
stdio.writeln()
```

m	*sample size*
n	*range*
perm[]	*permutation of 0 to n-1*

Initialization

Algorithm

Printing

Without Replacement

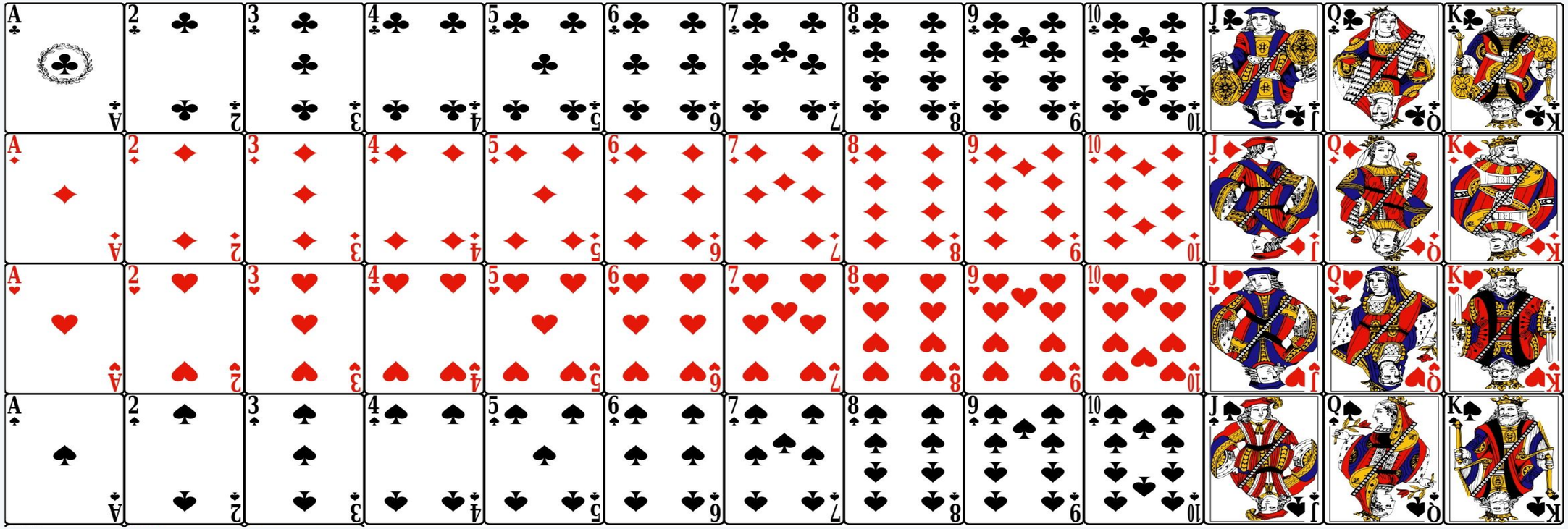
i	r	perm[]															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	9	9	1	2	3	4	5	6	7	8	0	10	11	12	13	14	15
1	5	9	5	2	3	4	1	6	7	8	0	10	11	12	13	14	15
2	13	9	5	13	3	4	1	6	7	8	0	10	11	12	2	14	15
3	5	9	5	13	1	4	3	6	7	8	0	10	11	12	2	14	15
4	11	9	5	13	1	11	3	6	7	8	0	10	4	12	2	14	15
5	8	9	5	13	1	11	8	6	7	3	0	10	4	12	2	14	15
		9	5	13	1	11	8	6	7	3	0	10	4	12	2	14	15

Trace of ***python sample.py 6 16***

```
# Create a random sample of size m in perm[0..m).
for i in range(m):
    r = random.randrange(i, n)

    # Exchange perm[i] and perm[r].
    temp    = perm[r]
    perm[r] = perm[i]
    perm[i] = temp
```

Swap

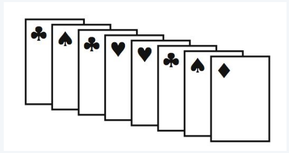
Coupon Collector



Coupon Collector Problem

Coupon collector problem. Given N different card types, how many do you have to collect before you have (at least) one of each type?

assuming each possibility is equally likely for each card that you collect



Simulation algorithm. Repeatedly choose an integer **i** between **0** and **N-1**.
Stop when we have at least one card of every type.

Q. How to check if we've seen a card of type **i**?
A. Maintain a boolean array so that **found[i]** is true if we've already collected a card of type **i**.

Coupon Collector: Implementation

Program 1.4.2 Coupon collector simulation (couponcollector.py)

```
import random
import sys
import stdarray
import stdio

n = int(sys.argv[1])

count = 0
collectedCount = 0
isCollected = stdarray.create1D(n, False)

while collectedCount < n:
    # Generate another coupon.
    value = random.randrange(0, n)
    count += 1
    if not isCollected[value]:
        collectedCount += 1
        isCollected[value] = True

stdio.writeln(count)
```

n	*# of coupon values (0 to n-1)*
count	*# of coupons collected*
collectedCount	*# of distinct coupons collected*
isCollected[i]	*has coupon i been collected?*
value	*value of current coupon*

type of next card (between 0 and N-1)

This program accepts an integer n as a command-line argument, and writes the number of coupons collected before obtaining one of each of n types. Thus it simulates a coupon collector.

Coupon Collector: Debugging

Debugging. Add code to print contents of all variables.

val	isCollected[] 0 1 2 3 4 5	count	collectedCount
	F F F F F F	0	0
2	F F T F F F	1	1
0	T F T F F F	2	2
4	T F T F T F	3	3
0	T F T F T F	3	4
1	T T T F T F	4	5
2	T T T F T F	4	6
5	T T T F T T	5	7
0	T T T F T T	5	8
1	T T T F T T	5	9
3	T T T T T T	6	10

Trace for a typical run of
python couponcollector.py 6

```
n = int(sys.argv[1])
count = 0
collectedCount = 0
isCollected = stdarray.create1D(n, False)

while collectedCount < n:
    # Generate another coupon.
    value = random.randrange(0, n)
    count += 1
    if not isCollected[value]:
        collectedCount += 1
        isCollected[value] = True

stdio.writeln(count)
```

Challenge. Debugging with arrays requires tracing many variables.

Coupon Collector: Mathematical Context

Coupon collector problem. Given N different possible cards, how many do you have to collect before you have (at least) one of each type?

Fact. About $N (1 + 1/2 + 1/3 + \ldots + 1/N) \sim N \ln N$.

Exercise: Coupon Collector

Estimate the output of the command python3 couponcollector.py 30 will print to the terminal if executed.

Answer

Estimate the output of the command python couponcollector.py 30 will print to the terminal if executed.

30xln(30) = 102

Coupon Collector: Scientific Context

Q. Given a sequence from nature, does it have same characteristics as a random sequence?

A. No easy answer - many tests have been developed.

Coupon collector test. Compare number of elements that need to be examined before all values are found against the corresponding answer for a random sequence.

Sieve of Eratosthenes

i	isPrime[]																							
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of **`python primesieve.py 25`**

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, …

Prime counting function. $\pi(N)$ = # primes ≤ N.
Ex. $\pi(17) = 7$.

Sieve of Eratosthenes.

- Maintain an array isPrime[] to record which integers are prime.
- Repeat for i=2 to N
  - if i is not still marked as prime
    i is is not prime since we previously found a factor
  - if i is marked as prime
    i is prime since it has no smaller factors
    mark all multiples of i to be non-prime

Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, …

Prime counting function. π(*N*) = # primes ≤ *N*.
Ex. π(25) = 9.

i	isPrime[]																							
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of **python primesieve.py 25**

Sieve of Eratosthenes

Program 1.4.3 Sieve of Eratosthenes (primesieve.py)

```
import sys
import stdarray
import stdio

n = int(sys.argv[1])

isPrime = stdarray.create1D(n+1, True)

for i in range(2, n):
    if (isPrime[i]):
        # Mark multiples of i as nonprime.
        for j in range(2, n//i + 1):
            isPrime[i*j] = False

# Count the primes.
count = 0
for i in range(2, n+1):
    if (isPrime[i]):
        count += 1
stdio.writeln(count)
```

Initialization

Sieve of Eratosthenes

Counting

Sieve of Eratosthenes

i	*isPrime[]*																							
	2	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of **python primesieve.py 25**

```
for i in range(2, n):
    if (isPrime[i]):
        # Mark multiples of i as nonprime.
        for j in range(2, n//i + 1):
            isPrime[i*j] = False
```

Exercise: Sieve of Eratosthenes

Compute π(20)

i	isPrime																						
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

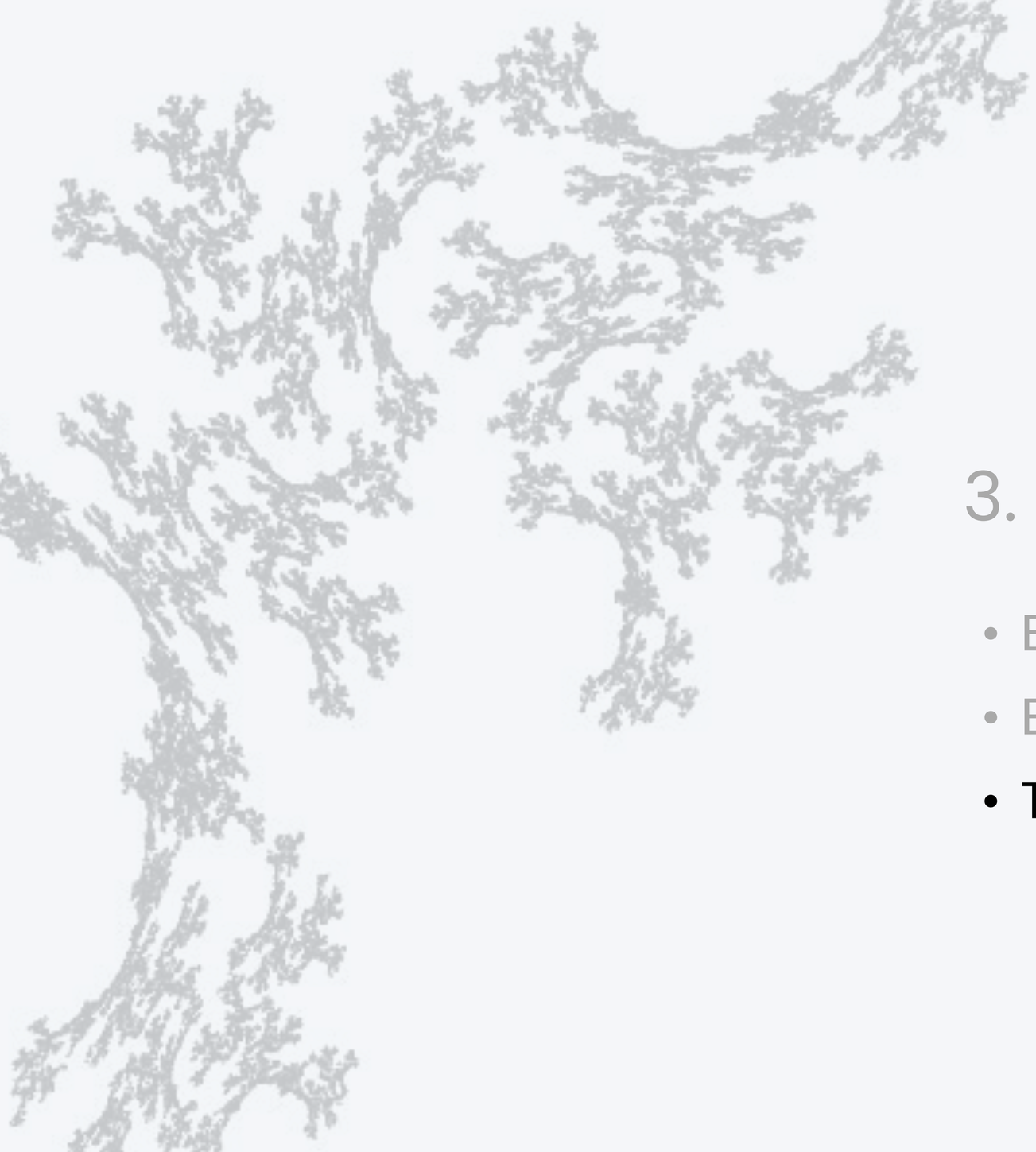
Trace of java PrimeSieve 25

Answer

$\pi(20) = 8$

i	isPrime																							
	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T
2	T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
3	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T
5	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F
	T	T	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	T	F	F	F	T	F	F

Trace of java PrimeSieve 25



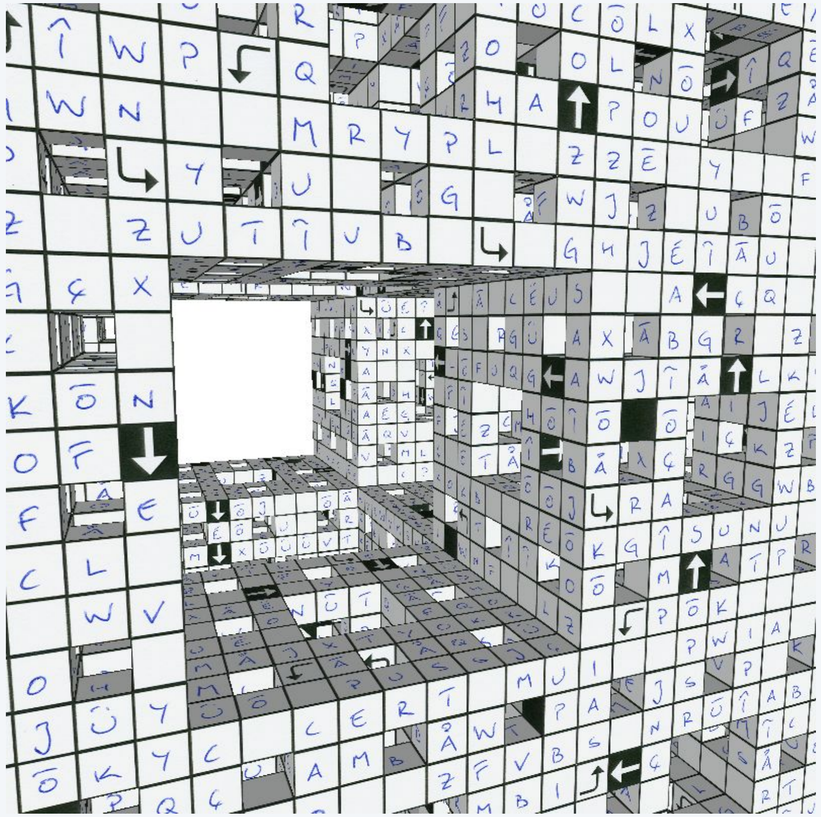
INTRO TO PROGRAMMING IN PYTHON

SEDGEWICK • WAYNE • DONDERO

3. Arrays

- Basic concepts
- Examples of array-processing code
- Two-dimensional arrays

Multidimensional Arrays



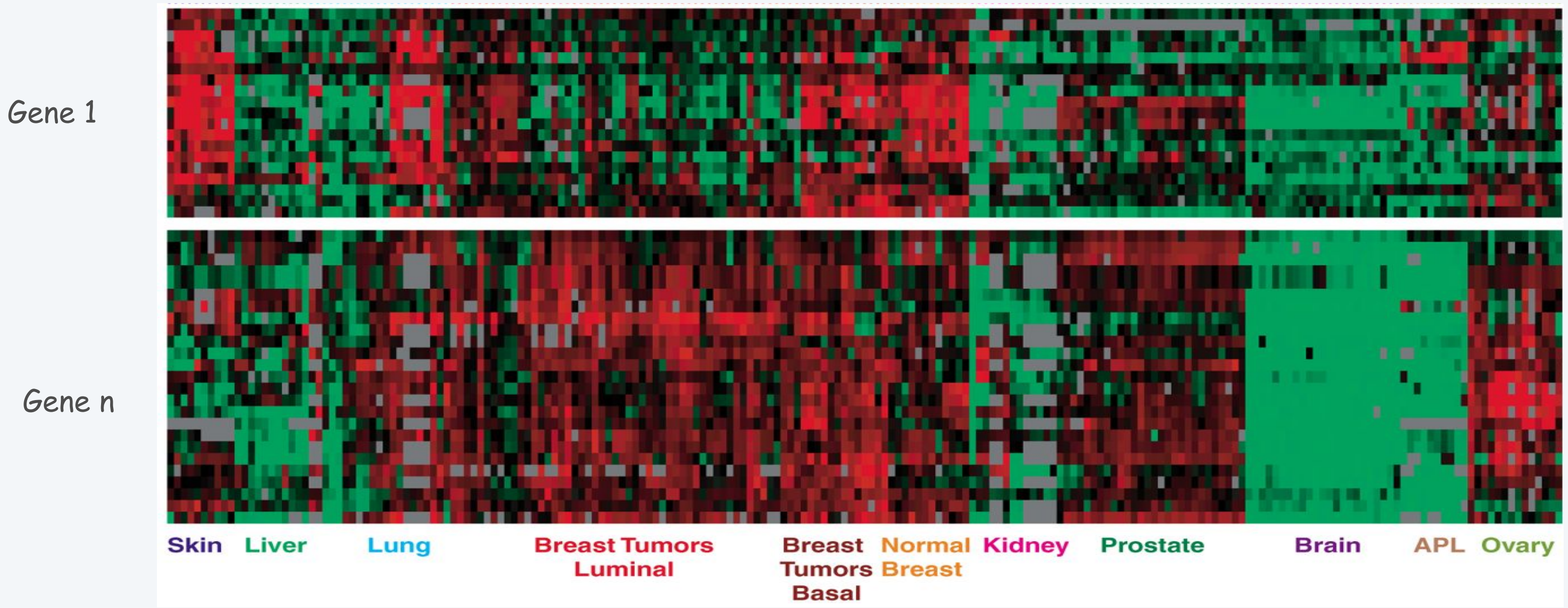
Two-Dimensional Arrays

Two-dimensional arrays.

- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction. Matrix.

Python abstraction. 2D array.



Reference: Botstein & Brown group

Python Support for 2D Arrays

Conceptually a two-dimensional array is an array of arrays.

operation	*typical code*
Create and Initialize	`a = [[0.30, 0.60, 0.10],[0.2, 0.2, 0.6]]`

Create and default Initialize	`a = []`
	`for i in range(m):`
	`  row = [0.0]*n`
	`  a += [row]`

Assign value	`a[2][0] = 2.17`
Access value	`v = a[5][2] # 5th row and 2nd column`

Iterate array	`for i in range(m):`
	`  for j in range(n):`
	`  v = a[i][j]`
	`  stdio.writeln( str(v) + ' ')`

BUT cost of creating an array is proportional to m x n.

Alternative without indices

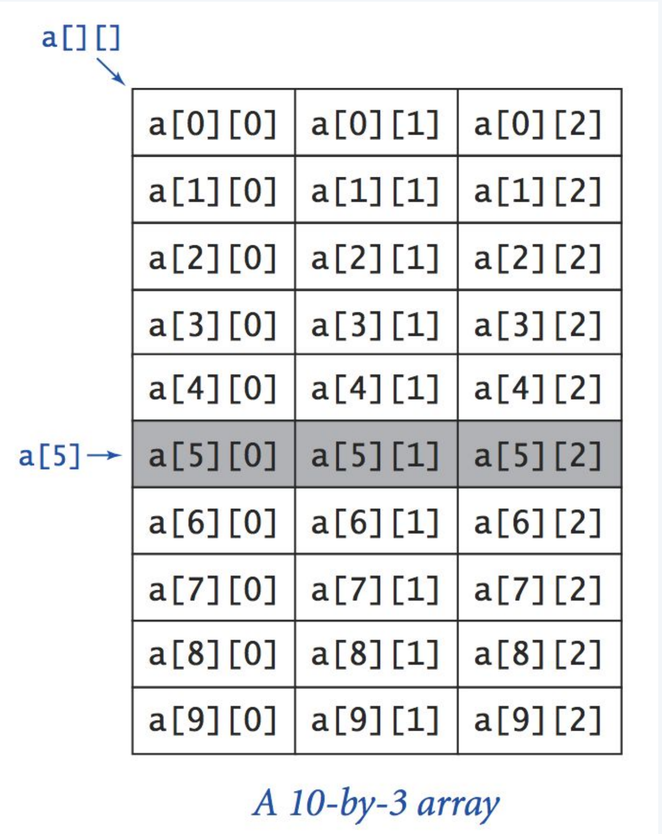
```
for row in a:
   for v in row:
      stdio.writeln(str(v)+' ')
```

Indexing 2D Arrays

Array access. Use **a[i][j]** to access entry in row **i** and column **j**.

Zero-based indexing. Row and column indices start at **0**.

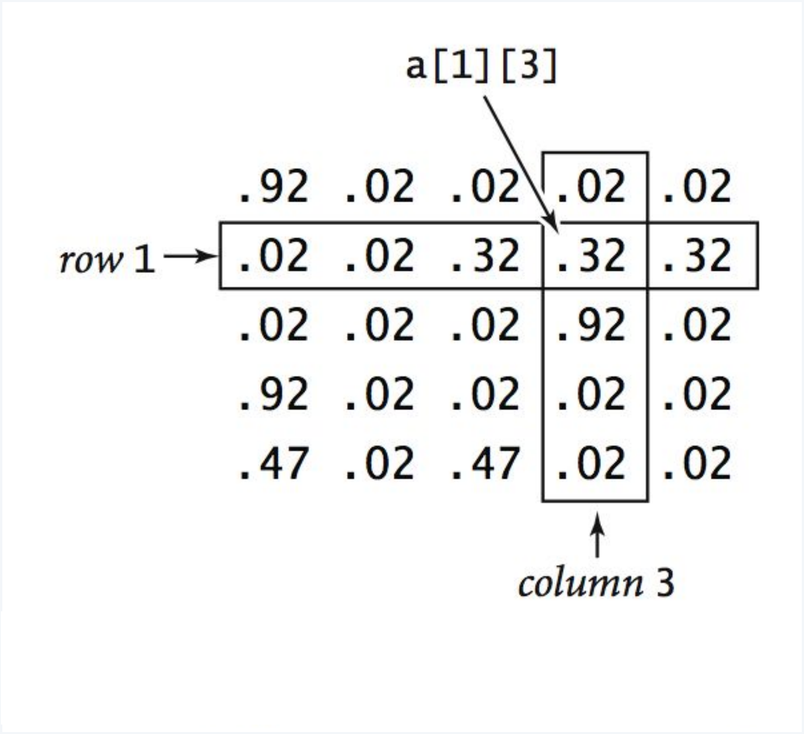
```
M = 10
N = 3
a = []

for i in range(M):
    row = [0.0]*N
    a += [row]

a = stdarray.create2D(M,N,0.0)
```



A 10-by-3 array

Setting 2D Array Values at Compile Time

```
p = [[ .92, .02, .02, .02, .02 ],
     [ .02, .02, .32, .32, .32 ],
     [ .02, .02, .02, .92, .02 ],
     [ .92, .02, .02, .02, .02 ],
     [ .47, .02, .47, .02, .02 ]]
```



Ragged 2D Arrays

Python allows each row array to have a different type and a different length.

Python allows different types and length for each array

```
# Create array of None
a = [None]*5     # len(m) = 5
a[0] = [True]*4 # bools  len(a[0]) = 4
a[1] = [0.0]*2  # floats len(a[1]) = 2
a[2] = ['NA']*3 # strs   len(a[2]) = 3
stdio.writeln(a)
>>>[[True, True, True, True], [0.0,
0.0, 0.0], ['NA', 'NA'], None, None]
```

can be different for each row

a[0] →	a[0][0] = True	a[0][1] = True	a[0][2] = True	a[0][3] = True
a[1] →	a[1][0] = 0.0	a[1][1] = 0.0		
a[2] →	a[2][0] = 'NA'	a[2][1] = 'NA'	a[2][2] = 'NA'	
a[3] = None				
a[4] = None				

ragged 2D array

Table view of 5-by-mixed length array

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector

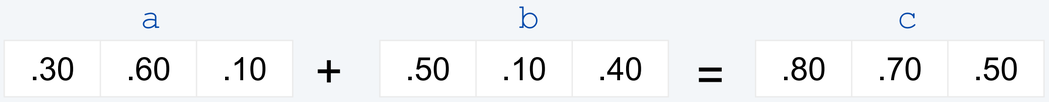
Python implementation: 1D array

Mathematical abstraction: matrix

Python implementation: 2D array

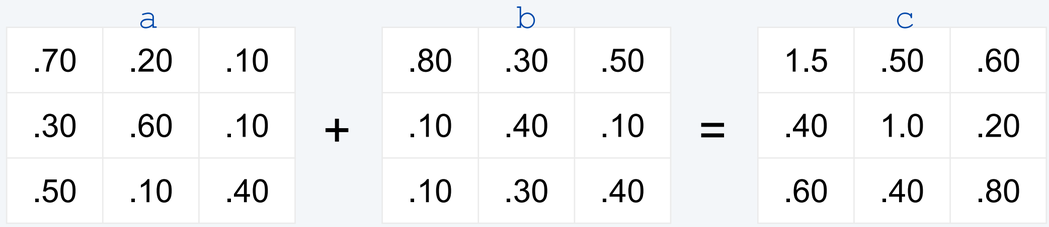
Vector addition

```
c = stdarray.create1D(N,0.0)
for i in range(N):
   c[i] = a[i] + b[i]
```



Matrix addition

```
c = stdarray.create2D(N,N,0.0)
for i in range(N):
   for j in range(N):
      c[i][j] = a[i][j] + b[i][j]
```



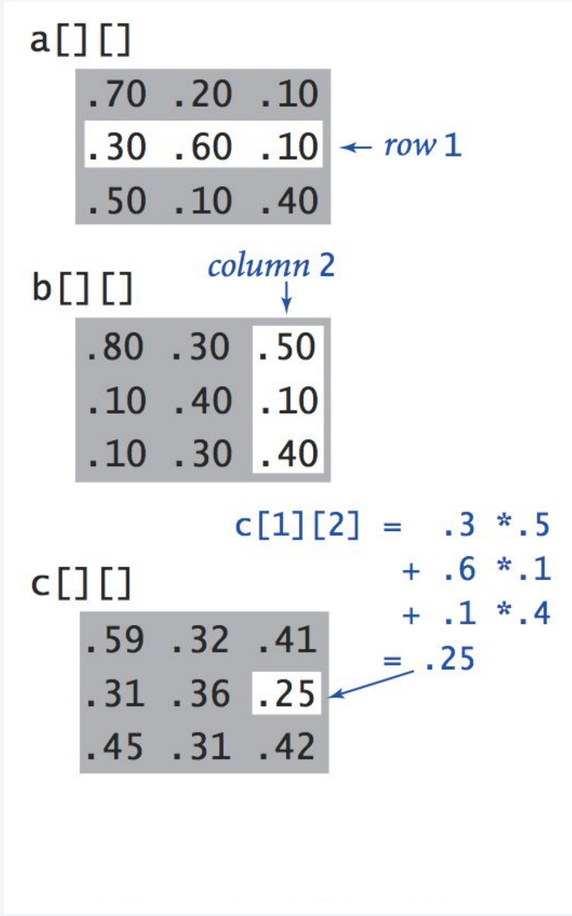
Matrix Multiplication

Matrix multiplication. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the dot product of the i^{th} row of `a[][]` and the j^{th} column of `b[][]`.

all values initialized to 0.0

```
c = stdarray.create2D(N,N,0.0)

for i in range(0,N):
    for j in range(0,N):
        for k in range(0,N):
            c[i][j] += a[i][k] * b[k][j]
```

dot product of row `i` of `a[][]` and column `j` of `b[][]`



Exercise: Matrix Multiplication

Compute the following matrix products?

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}^2 = ?$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} = ?$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Answer

Compute the following matrix products? (**NOT COMMUTATIVE**)

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix}^2 = \begin{bmatrix} 6 & 12 & 18 \\ 6 & 12 & 18 \\ 6 & 12 & 18 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 3 & 10 & 13 \\ 3 & 10 & 13 \\ 3 & 10 & 13 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \\ 1 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 6 & 6 \\ 8 & 8 & 8 \\ 12 & 12 & 12 \end{bmatrix}$$

Exercise: Scalar multiplications

Q. How many scalar multiplications multiply two N-by-N matrices?

A. N B. N^2 C. N^3 D. N^4

```
c = stdarray.create2D(N,N,0.0)

for i in range(0,N):
    for j in range(0,N):
        for k in range(0,N):
            c[i][j] += a[i][k] * b[k][j]
```

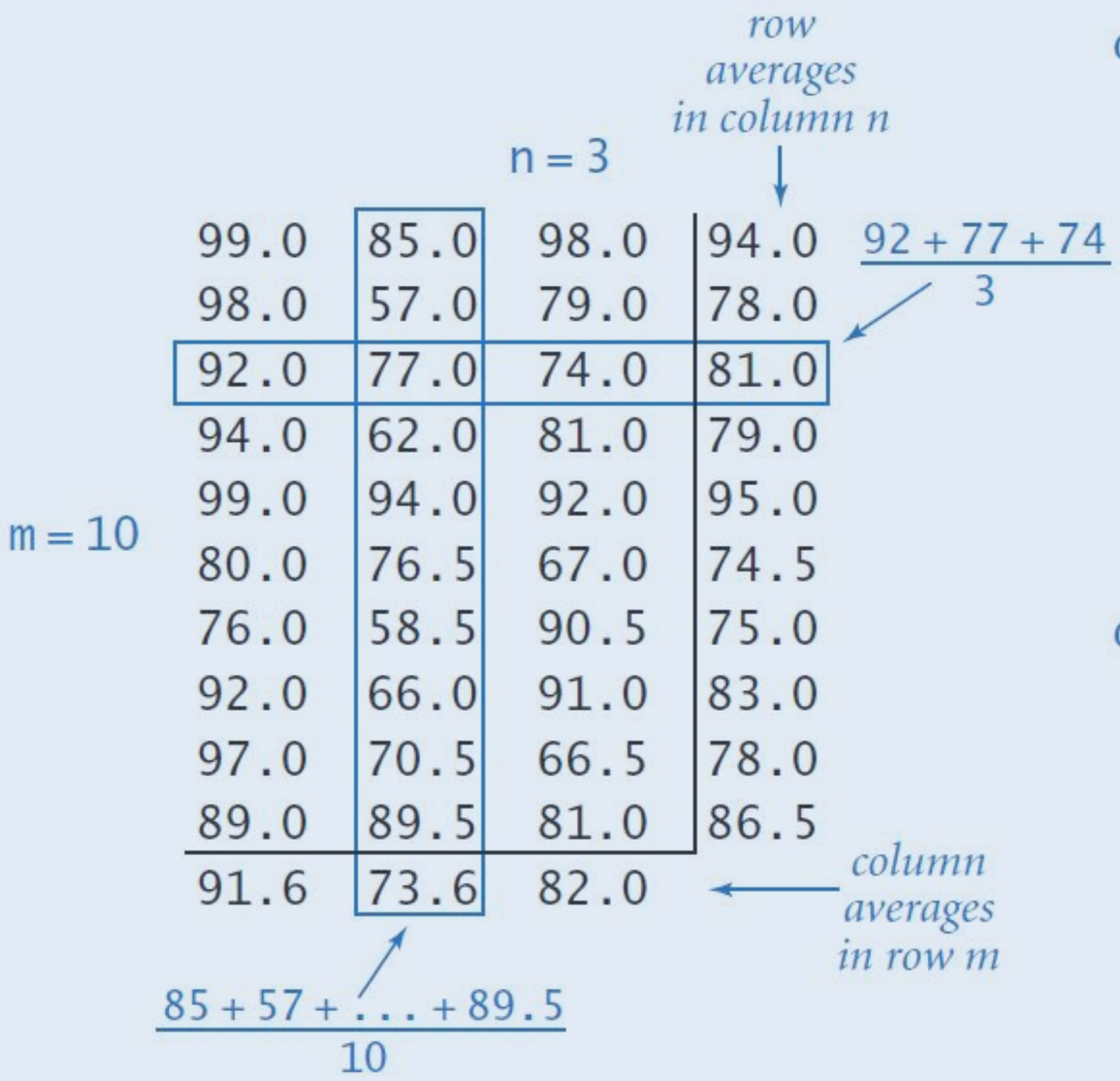
Answer

Q. How many scalar multiplications multiply two N-by-N matrices?

A. N B. N^2 C. N^3 D. N^4

```
c = stdarray.create2D(N,N,0.0)

for i in range(0,N):
    for j in range(0,N):
        for k in range(0,N):
            c[i][j] += a[i][k] * b[k][j]
```

Averages



Compute row averages (row-major order)

```
for i in range(m):
    # Average for row i
    total = 0.0
    for j in range(n):
        total += a[i][j]
    a[i][n] = total / m
```

Compute column averages (column-major order)

```
for j in range(n):
    # Average for column j
    total = 0.0
    for i in range(m):
        total += a[i][j]
    a[m][j] = total / n
```

Typical spreadsheet calculations

Vector-matrix products

Matrix–vector multiplication `b[] = a[][]*x[]`

```
b = stdarray.create1D(m, 0.0)
for i in range(m):
   for j in range(n):
      b[i] += a[i][j]*x[j]
```

dot product of row i of a[][] and x[]

a[][]

99.0	85.0	98.0
98.0	57.0	79.0
92.0	77.0	74.0
94.0	62.0	81.0
99.0	94.0	92.0
80.0	76.5	67.0
76.0	58.5	90.5
92.0	66.0	91.0
97.0	70.5	66.5
89.0	89.5	81.0

x[]

0.33333
0.33333
0.33333

b[]

94.0
78.0
81.0
79.0
95.0
74.5
75.0
83.0
78.0
86.5

← *row averages*

Vector–matrix multiplication `c[] = y[]*a[][]`

```
c = stdarray.create1D(n, 0.0)
for j in range(n):
   for i in range(m):
      c[j] += y[i]*a[i][j]
```

dot product of column j of a[][] and y[]

y[] [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1]

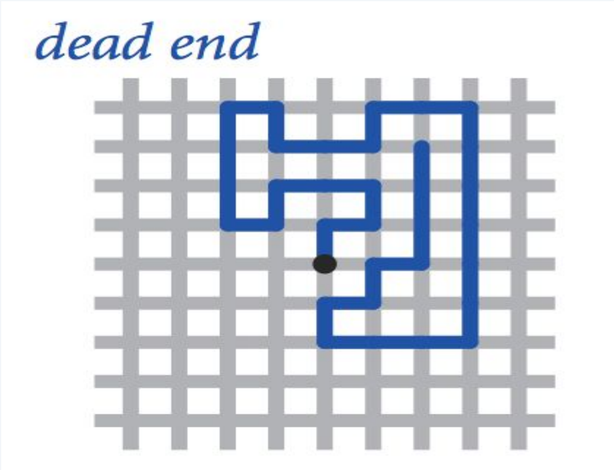
a[][]

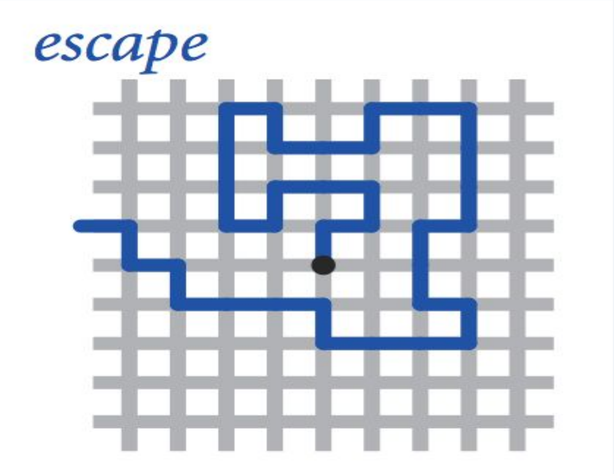
99.0	85.0	98.0
98.0	57.0	79.0
92.0	77.0	74.0
94.0	62.0	81.0
99.0	94.0	92.0
80.0	76.5	67.0
76.0	58.5	90.5
92.0	66.0	91.0
97.0	70.5	66.5
89.0	89.5	81.0

c[] [91.6 73.6 82.0] ← *column averages*

Special cases of matrix operations (when one of the arguments is a vector)

Self-Avoiding Walk

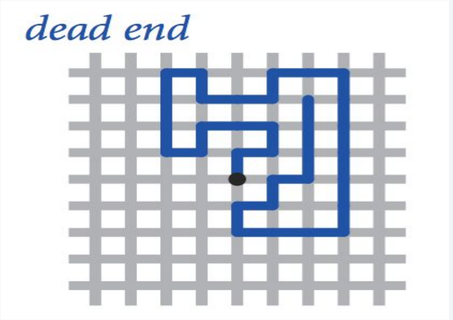


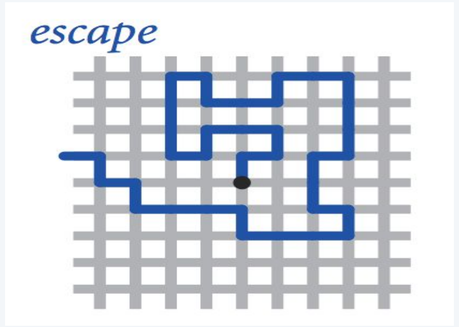


Self-Avoiding Walk

Model.

- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.
- Two possible outcomes: dead end and escape.



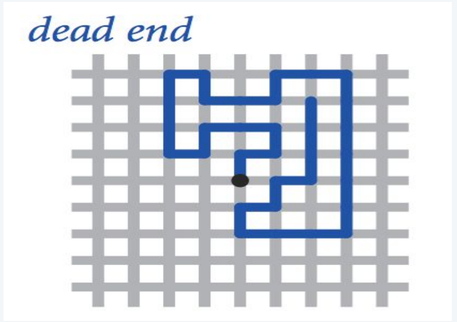


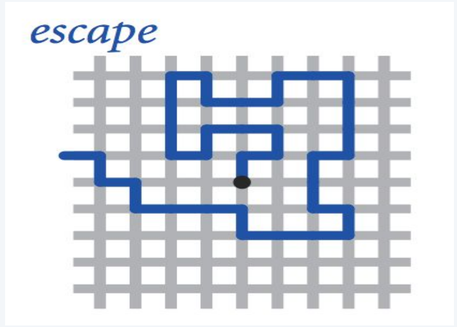
Applications. Polymers, statistical mechanics, etc.

Q. What fraction of time will you escape in an 5-by-5 lattice?
Q. In an N-by-N lattice?

Exercise: Skeleton

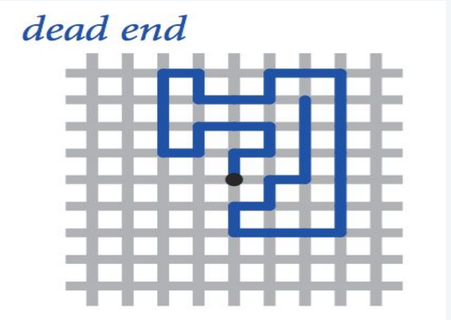
Skeleton. Before writing any code, write comments to describe what you want your program to do.

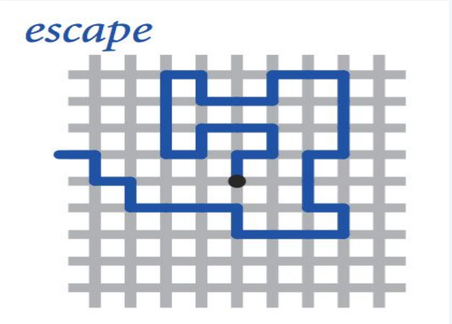




Answer

Skeleton. Before writing any code, write comments to describe what you want your program to do.






```
// imports
def main():
    // Read in lattice size N as command-line argument.
    // Read in number of trials T as command-line argument.

    // Repeat T times:

        // Initialize (x, y) to center of N-by-N grid.

        // Repeat as long as (x, y) stays inside N-by-N grid:
            // Check for dead end and update count.
            // Mark (x, y) as visited.
            // Take a random step, updating (x, y).

    // Print fraction of dead ends
```

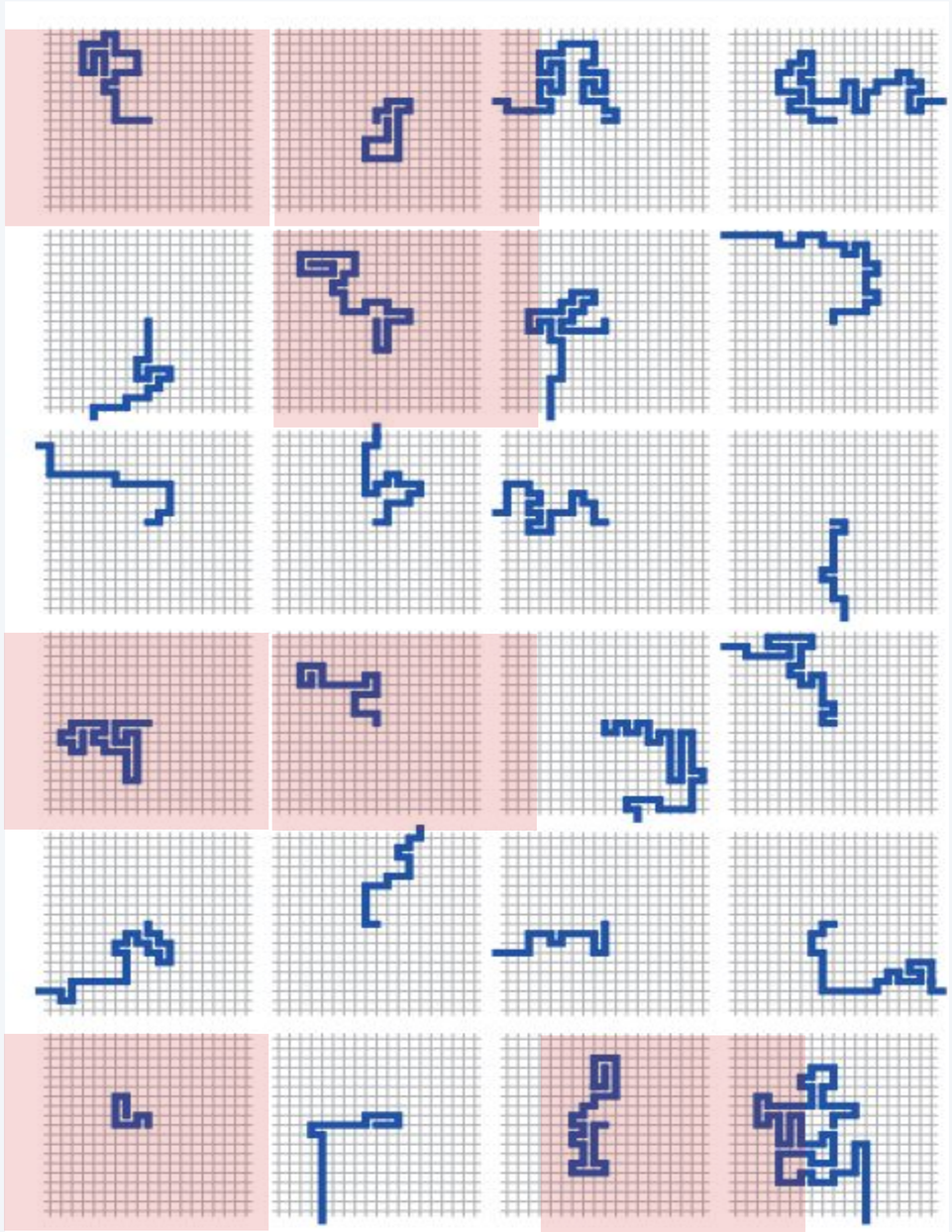
how to implement?

Self-Avoiding Walk: Implementation

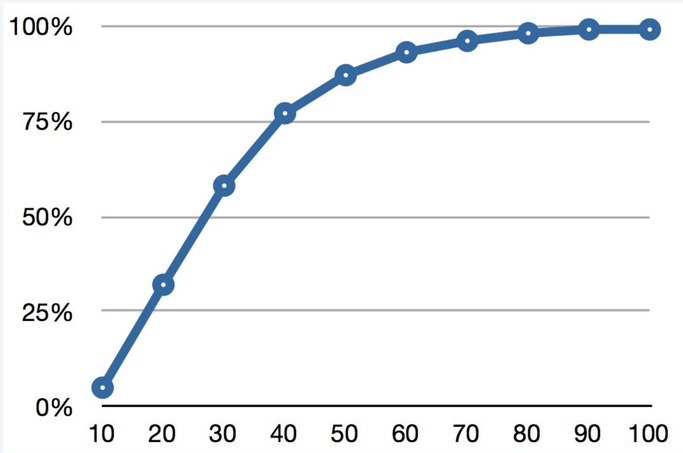
```
import random
import sys
import stdarray
import stdio
import stdrandom

def main():
    N      = int(sys.argv[1])
    trials = int(sys.argv[2])
    deadEnds = 0
    for t in range(trials):
        a = stdarray.create2D(N,N,False)
        x = N//2
        y = N//2
        while (x > 0) and (x < N-1) and (y > 0) and (y < N-1):
            # Check for dead end and make random move.
            a[x][y] = True
            if (a[x-1][y] and a[x+1][y] and a[x][y-1] and a[x][y+1]):
                deadEnds += 1
                break
            r = stdrandom.uniformInt(1,5)
            if   (r == 1) and (not a[x+1][y]): x += 1
            elif (r == 2) and (not a[x-1][y]): x -= 1
            elif (r == 3) and (not a[x][y+1]): y += 1
            elif (r == 4) and (not a[x][y-1]): y -= 1
    stdio.writeln(str(100*deadEnds//trials) + '% dead ends')

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

only take step if site is unoccupied

Visualization of Self-Avoiding Walks



```
% python selfavoid.py 10 100000
5% dead ends

% python selfavoid.py 20 100000
32% dead ends

% python selfavoid.py 30 100000
58% dead ends
…

% python selfavoid.py 100 100000
99% dead ends
```

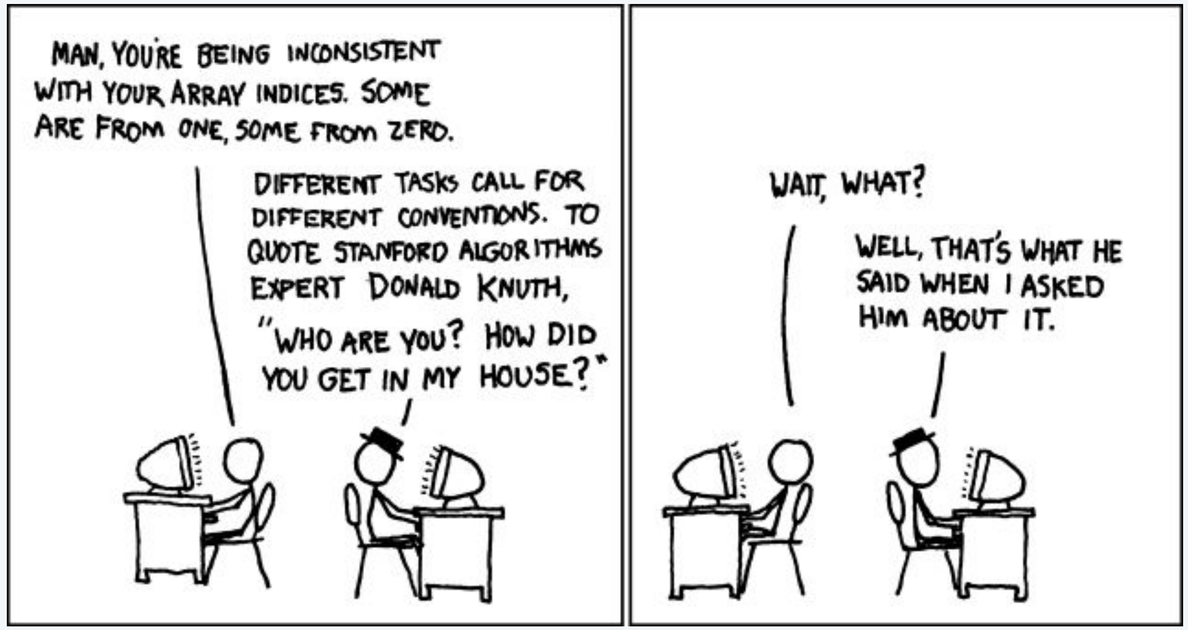


Summary

Arrays.

- Organized way to store huge quantities of data.
- Almost as easy to use as primitive types.
- Can directly access an element given its index.

Ahead. Reading in large quantities of data from a file into an array.



http://imgs.xkcd.com/comics/donald_knuth.png