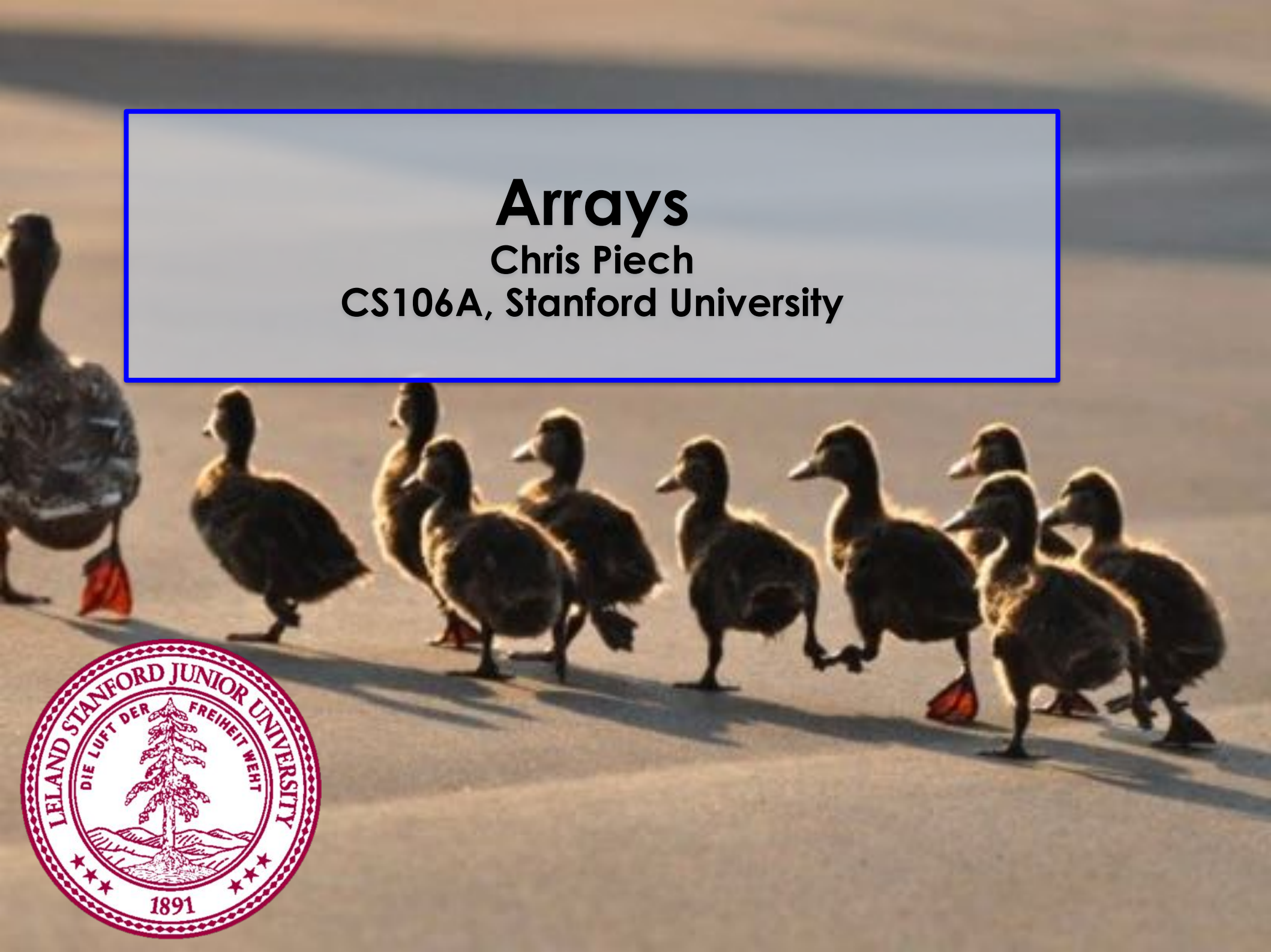
Arrays

Chris Piech
CS106A, Stanford University



What does this say?

```
53‡‡†305))6*;4826)4‡•)4‡);806*;48†8¶
60))85;1‡(;:‡*8†83(88)5*†;46(;88*96*
?;8)*‡(;485);5*†2:*‡(;4956*2(5*-4)8¶
8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡
1;48†85;4)485†528806*81(‡9;48;(88;4(
‡?34;48)4‡;161;:188;‡?;
```

Puzzle in Gold Bug by Edgar Allan Poe

Changing Variable Types

int to double?

```
int x = 5;
double xDbl = x;
```

int to String?

```
int x = 5;
String xStr = "" + x
```

String to int?

```
String xStr = "5";
int x = Integer.parseInt(x);
```

String to double?

```
String xStr = "5.6";
double x = Double.parseDouble(xStr);
```

Casting double to int

```
double x = 5.2;
int y = (int)x;
```

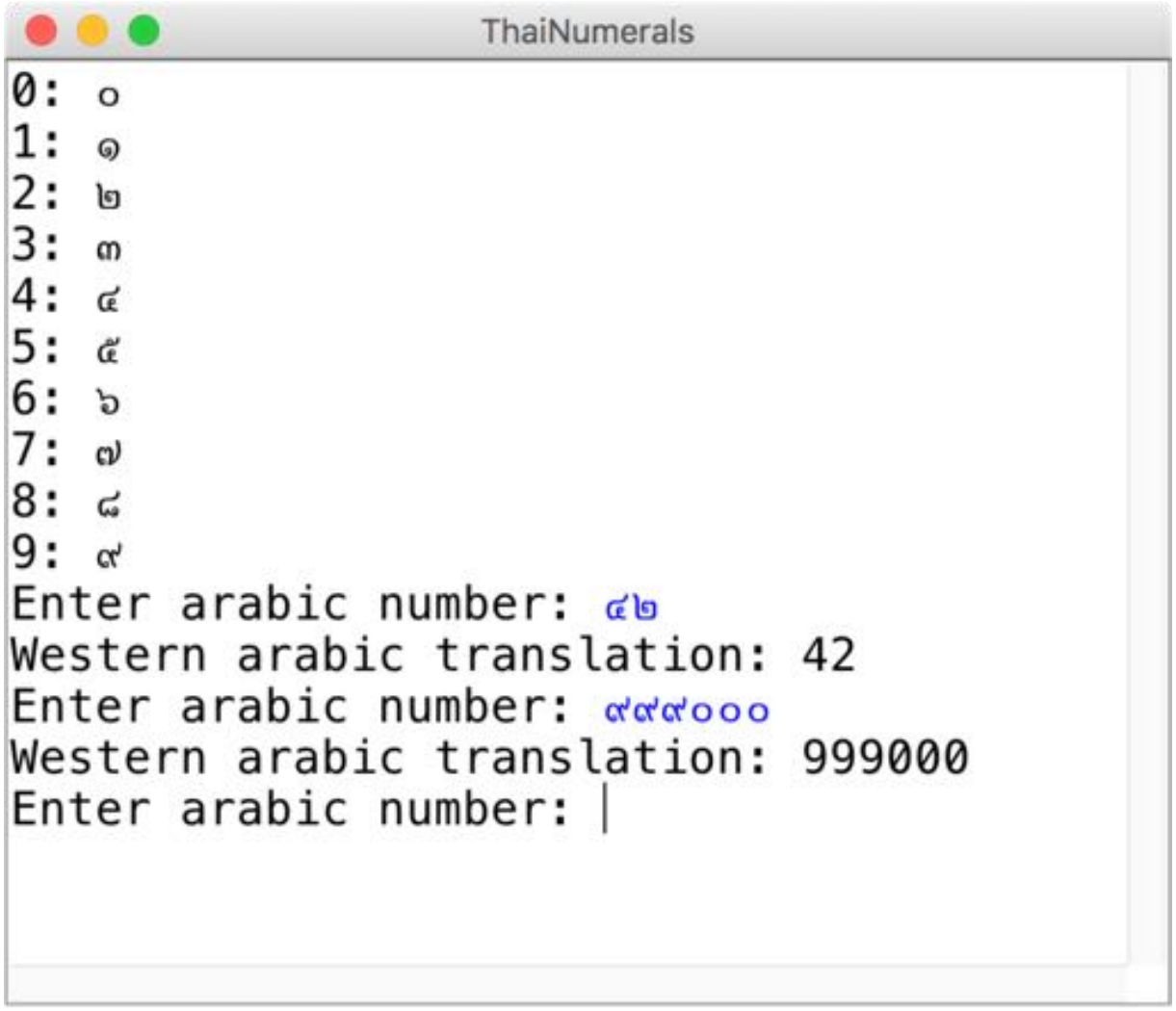
GObject to GRect

```
GObject obj = getElementAt(5, 2);
GRect objRect = (GRect)obj;
```

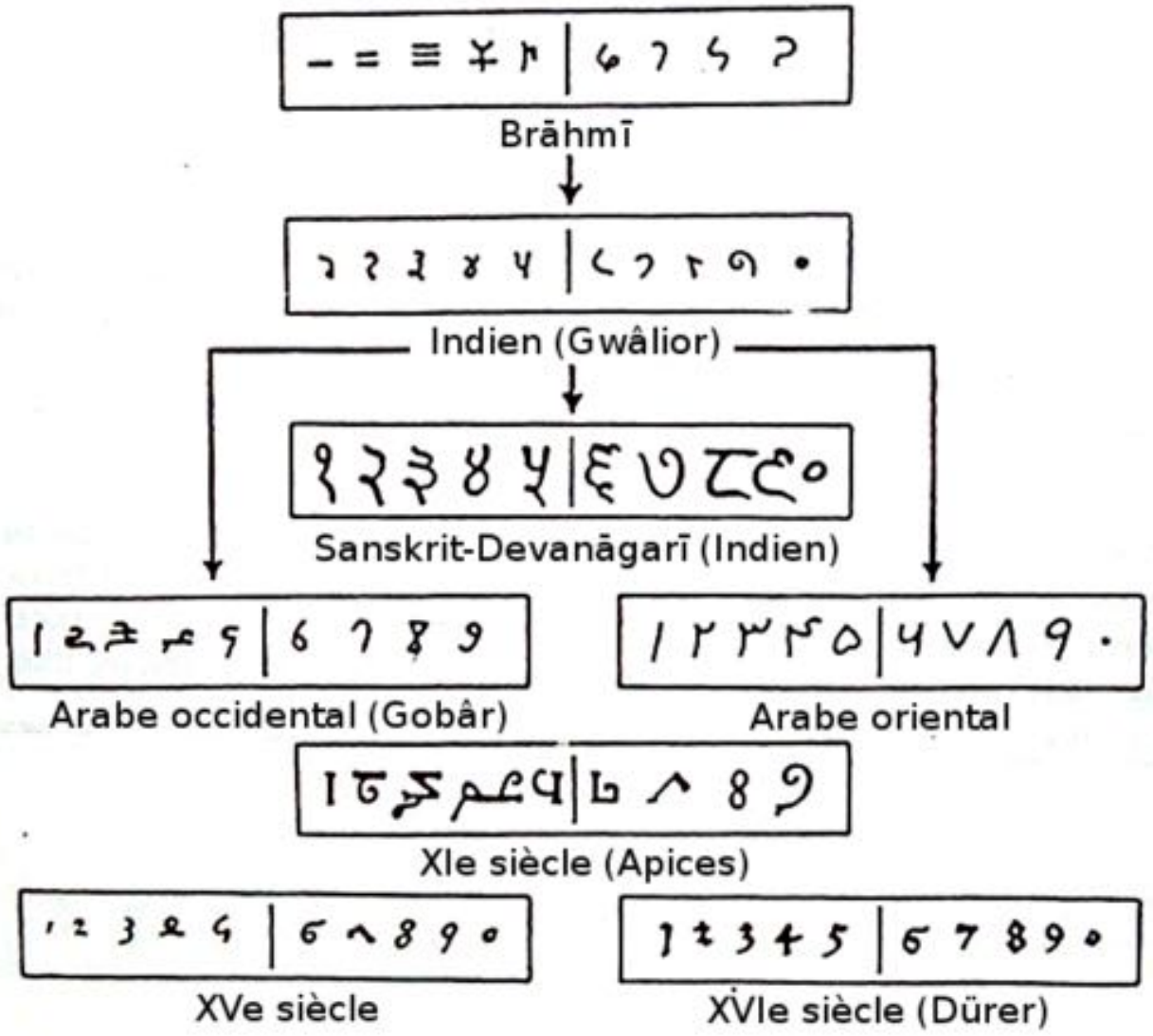
int to char

```
int diff = 'C'-'A';
char next = (char)'a' + diff;
```

Changing Variable Types

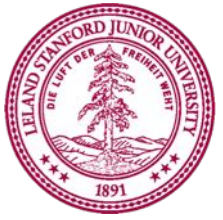


Number Translation



Where are we?

- Karel the Robot
- Java
- Console Programs
- Graphics Programs
- Text Processing
- **Data Structures**
- Defining our own Variable Types
- GUIs

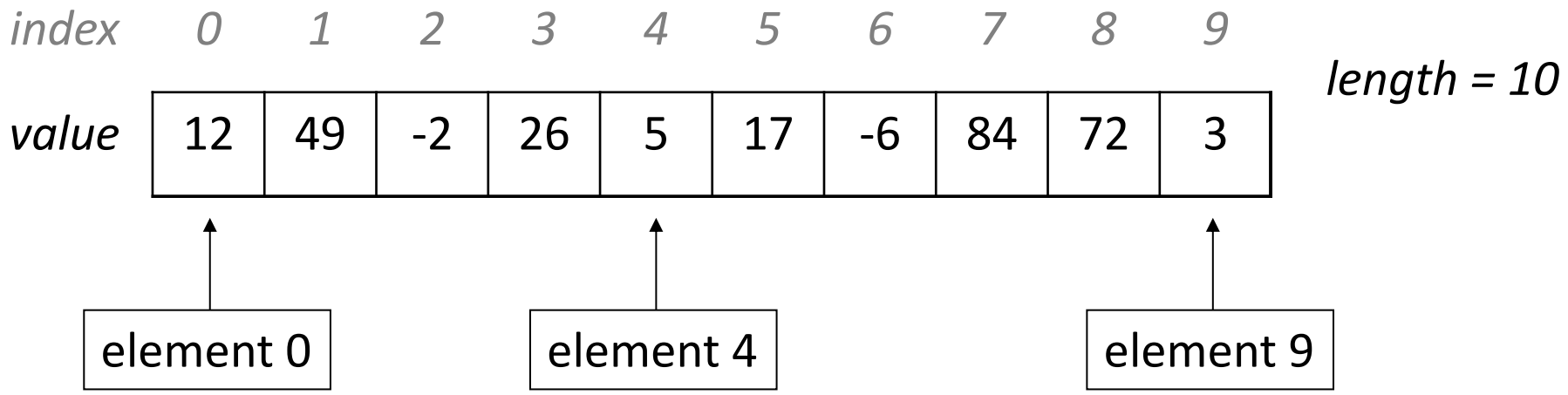


Arrays

#majorkey of the day

A new variable type that is an object that represents an ordered, homogeneous list of data.

– Arrays have many *elements* that you can access using *indices*



Many flavors of arrays

You can create arrays of any variable type. For example:

```
double[] results = new double[5];

String[] names = new String[3];

boolean[] switches = new boolean[4];

GRect[] rects = new GRect[5];
```

- Java initializes each element of a new array to its *default value*, which is **0** for `int` and `double`, '\0' for `char`, **`false`** for `boolean`, and **`null`** for objects.

Many flavors of arrays

You can create arrays of any variable type. For example:

```
char[] oldSchoolString = new char[5];
```

- Java initializes each element of a new array to its *default value*, which is **`0`** for `int` and `double`, **`'\0'`** for `char`, **`false`** for `boolean`, and **`null`** for objects.

Data Structures

Operation	Strings	Arrays
Make a new one	`String str = "abc";`	
Get length?	`str.length()`	
Get element?	`str.charAt(i)`	
Set element?	*Not allowed*	
Loop?	`for(int i = 0; i < str.length(); i++)`	

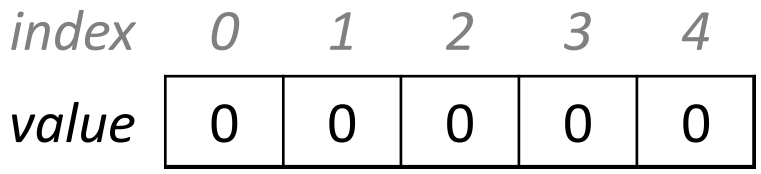
Data Structures

Operation	Strings	Arrays
Make a new one	`String str = "abc";`	`int arr = new int[5];`
Get length?	`str.length()`	`arr.length`
Get element?	`str.charAt(i)`	`arr[i]`
Set element?	*Not allowed*	`arr[i] = 5;`
Loop?	`for(int i = 0; i < str.length(); i++)`	`for(int i = 0; i < arr.length(); i++)`

Creating Arrays

```
type[] name = new type[length];
```

```
int[] numbers = new int[5];
```



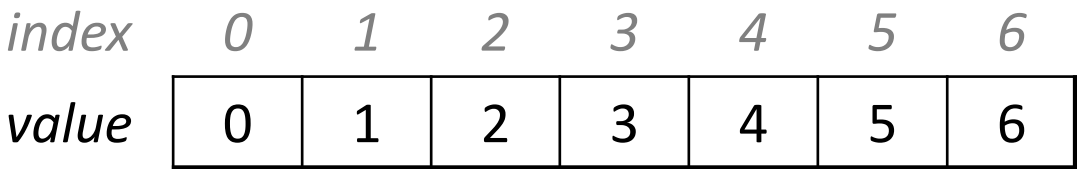
Java automatically initializes elements to **0**.

Getting values

```
name[index]    // get element at index
```

- Like Strings, indices go from **0** to the **array's length - 1**.

```
for (int i = 0; i < 7; i++) {
    println(numbers[i]);
}
println(numbers[9];    // exception
println(numbers[-1]);  // exception
```



Setting values

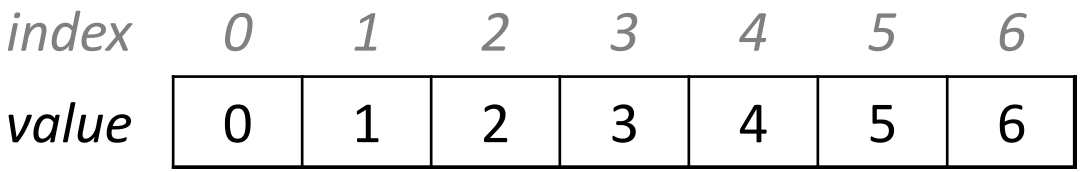
```
name[index] = value;    // set element at index
```

Setting values

```
name[index] = value;    // set element at index
```

- Like Strings, indices go from **0** to the **array's length - 1**.

```
int[] numbers = new int[7];
for (int i = 0; i < 7; i++) {
    numbers[i] = i;
}
numbers[8] = 2;    // exception
numbers[-1] = 5;   // exception
```



Practice

Q: What are the contents of `numbers` after executing this code?

```
int[] numbers = new int[8];
numbers[1] = 3;
numbers[4] = 7;
numbers[6] = 5;
int x = numbers[1];
numbers[x] = 2;
numbers[numbers[4]] = 9;
```

```
//  0  1  2  3  4  5  6  7
```

A. {0, 3, 0, 2, 7, 0, 5, 9}
B. {0, 3, 0, 0, 7, 0, 5, 0}
C. {3, 3, 5, 2, 7, 4, 5, 0}
D. {0, 3, 0, 2, 7, 6, 4, 4}

Getting “length”

Similar to a String, you can get the length of an array by saying

```
myArray.length
```

Note that there are *no parentheses* at the end!

Practice:

- What is the index of the *last element* of an array in terms of its length?
- What is the index of the *middle element* of an array in terms of its length?

Arrays loops

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

Arrays loops

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can efficiently initialize arrays.

```
int[] numbers = new int[8];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = 2 * i;
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
value	0	2	4	6	8	10	12	14

Arrays loops

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

For instance, we can read in numbers from the user:

```
int length = readInt("# of numbers? ");
int[] numbers = new int[length];
for (int i = 0; i < numbers.length; i++) {
    numbers[i] = readInt("Elem " + i + ": ");
}
```

Arrays loops

Just like with Strings, we can use an array's length, along with its indices, to perform cool operations.

Try it out! *sum up* all of an array's elements.

```
// assume that the user has created int[] numbers
int sum = 0;
for (int i = 0; i < numbers.length; i++) {
    sum += numbers[i];
}
println(sum);
```

Annoying initialization

Sometimes, we want to hardcode the elements of an array.

```
int numbers = new int[7];
numbers[0] = 5;
numbers[1] = 32;
numbers[3] = 12;
...

// This is tedious!
```

Fancy initialization

Sometimes, we want to hardcode the elements of an array.

Luckily, Java has a special syntax for initializing arrays to hardcoded numbers.

```
type[] name = { elements };
```

```
// Java infers the array length
int[] numbers = {5, 32, 12, 2, 1, -1, 9};
```

Limitations of Arrays

- An array's length is **fixed**. You cannot resize an existing array:

```
int[] a = new int[4];
a.length = 10;                          // error
```

- You cannot compare arrays with == or `equals` :

```
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) {  ... }                  // false!
if (a1.equals(a2)) {  ... }             // false!
```

- An array does not know how to print itself:

```
println(a1);                          // [I@98f8c4]
```

Array Methods to the Rescue!

- The class `Arrays` in package `java.util` has useful methods for manipulating arrays:

Method name	Description
`Arrays.binarySearch(`***`array, value`***`)`	returns the index of the given value in a *sorted* array (or < 0 if not found)
`Arrays.copyOf(`***`array, length`***`)`	returns a new copy of array of given length
`Arrays.equals(`***`array1, array2`***`)`	returns `true` if the two arrays contain same elements in the same order
`Arrays.fill(`***`array, value`***`);`	sets every element to the given value
`Arrays.sort(`***`array`***`);`	arranges the elements into sorted order
`Arrays.toString(`***`array`***`)`	returns a string representing the array, such as `"[10, 30, -25, 17]"`

Array Methods to the Rescue!

`Arrays.toString` accepts an array as a parameter and returns a string representation of its elements.

```
int[] e = {0, 2, 4, 6, 8};
e[1] = e[3] + e[4];
println("e is " + Arrays.toString(e));
```

Output:

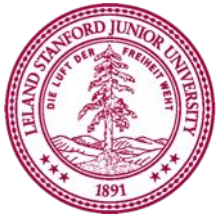
```
e is [0, 14, 4, 6, 8]
```

Arrays as Parameters

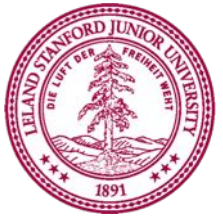
- Arrays are just another variable type, so methods can take arrays as parameters and return an array.

```
private int sumArray(int[] numbers) {
    ...
}

private int[] makeSpecialArray(...) {
    ...
    return myArray;
}
```

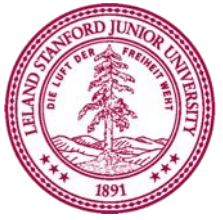



- Arrays are just another variable type, so methods can take arrays as parameters and return an array.
- However, arrays are **objects**, so per A Variable Origin Story, an array variable box actually stores its *location*.
- This means changes to an array passed as a parameter *affect the original array*!



```
public void run() {
    int[] numbers = new int[7];
    fillArray(numbers);
    println(Arrays.toString(numbers));
}

private void fillArray(int[] arr) {
    for (int i = 0; i < arr.length; i++) {
        arr[i] = 2 * i;
    }
}
```

Practice: Swapping Elements

Let's write a method called **swapElements** that swaps two elements of an array. How can we do this?

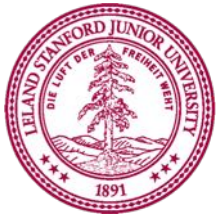
What parameters should it take (if any)? What should it return (if anything)?

```
private ??? swapElements(???) {
    ...
}
```


Swapping: Take 1

```
public void run() {
      int[] array = new int[5];
      ...
      swapElements(array[0], array[1]);
      ...
}

private void swapElements(int x, int y) {
      int temp = x;
      x = y;
      y = temp;
}
```

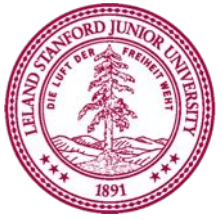



Swapping: Take 1

```
public void run() {
    int[] array = new int[5];
}

private void swapElements(int x, int y) {
    int temp = x;
    x = y;
    y = temp;
}
```

Ints are primitives, so they are passed by **value**! Their variable boxes store their *actual values*. So changes to the parameter *do not affect the original*.



Swapping: Take 2

```
public void run() {
      int[] array = new int[5];
      ...
      swapElements(array, 0, 1);
      ...
}


private void swapElements(int[] arr, int pos1, int pos2) {
      int temp = arr[pos1];
      arr[pos1] = arr[pos2];
      arr[pos2] = temp;
}
```


Swapping: Take 2

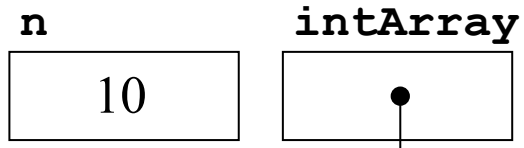
```
public void run() {
        int[] array = new int[5];
}
```

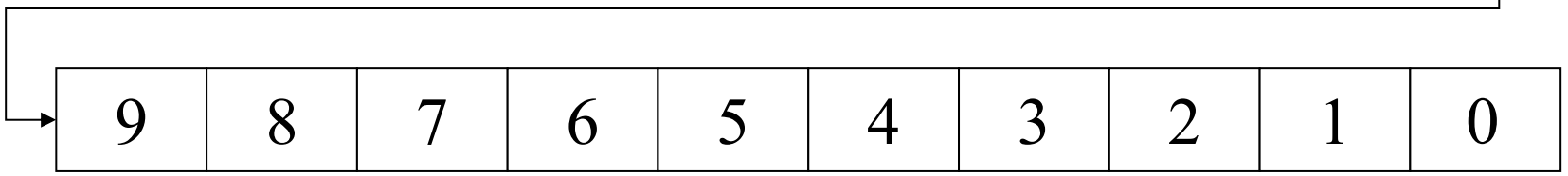
Arrays are **objects**, so they are passed by **reference**! Their variable boxes store their *location*. So changes to the parameter *do affect the original*.

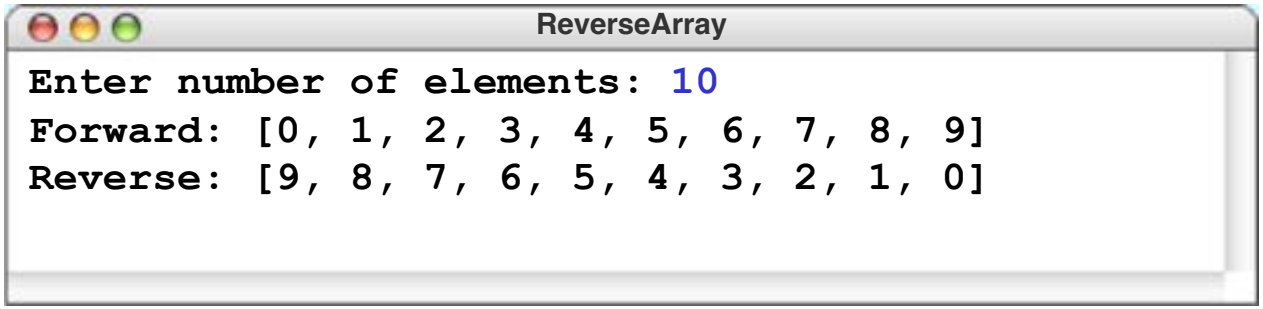
```
private void swapElements(int[] arr, int pos1, int pos2) {
        int temp = arr[pos1];
        arr[pos1] = arr[pos2];
        arr[pos2] = temp;
}
```

Example: Reverse Array Program

```
public void run() {
   int n = readInt("Enter number of elements: ");
   int[] intArray = createIndexArray(n);
   println("Forward: " + arrayToString(intArray));
   reverseArray(intArray);
   println("Reverse: " + arrayToString(intArray));
}
```



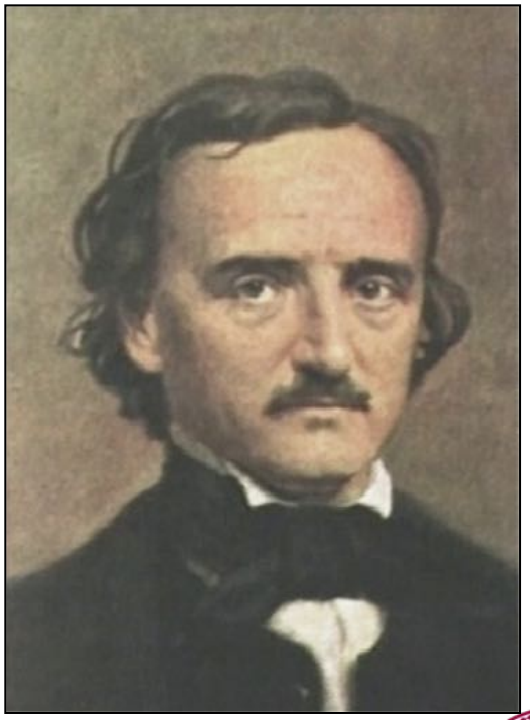




skip simulation

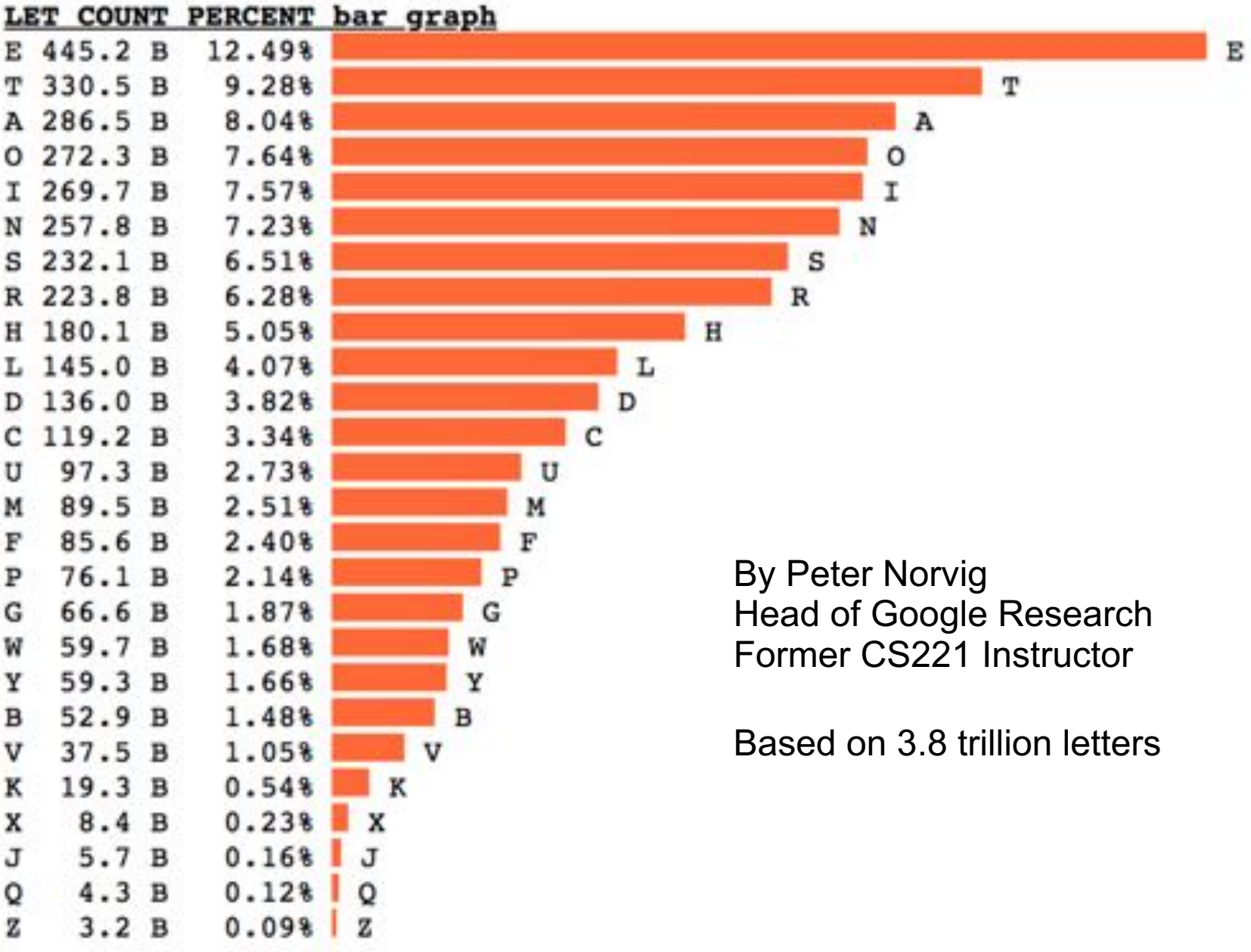
Cryptogram

- A ***cryptogram*** is a puzzle in which a message is encoded by replacing each letter in the original text with some other letter. The substitution pattern remains the same throughout the message. Your job in solving a cryptogram is to figure out this correspondence.
- One of the most famous cryptograms was written by Edgar Allan Poe in his short story "The Gold Bug."
- In this story, Poe describes the technique of assuming that the most common letters in the coded message correspond to the most common letters in English, which are E, T, A, O, I, N, S, H, R, D, L, and U.



Edgar Allan Poe (1809-1849)

Letter Frequency



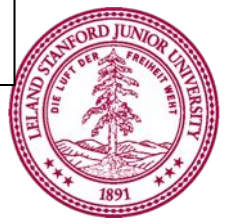
By Peter Norvig
Head of Google Research
Former CS221 Instructor

Based on 3.8 trillion letters

Poe's Cryptographic Puzzle

```
53‡‡†305))6*;4826)4‡•)4‡);806*;48†8¶
60))85;1‡(;:‡*8†83(88)5*†;46(;88*96*
?;8)*‡(;485);5*†2:*‡(;4956*2(5*–4)8¶
8*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡
1;48†85;4)485†528806*81(‡9;48;(88;4(
‡?34;48)4‡;161;:188;‡?;
```

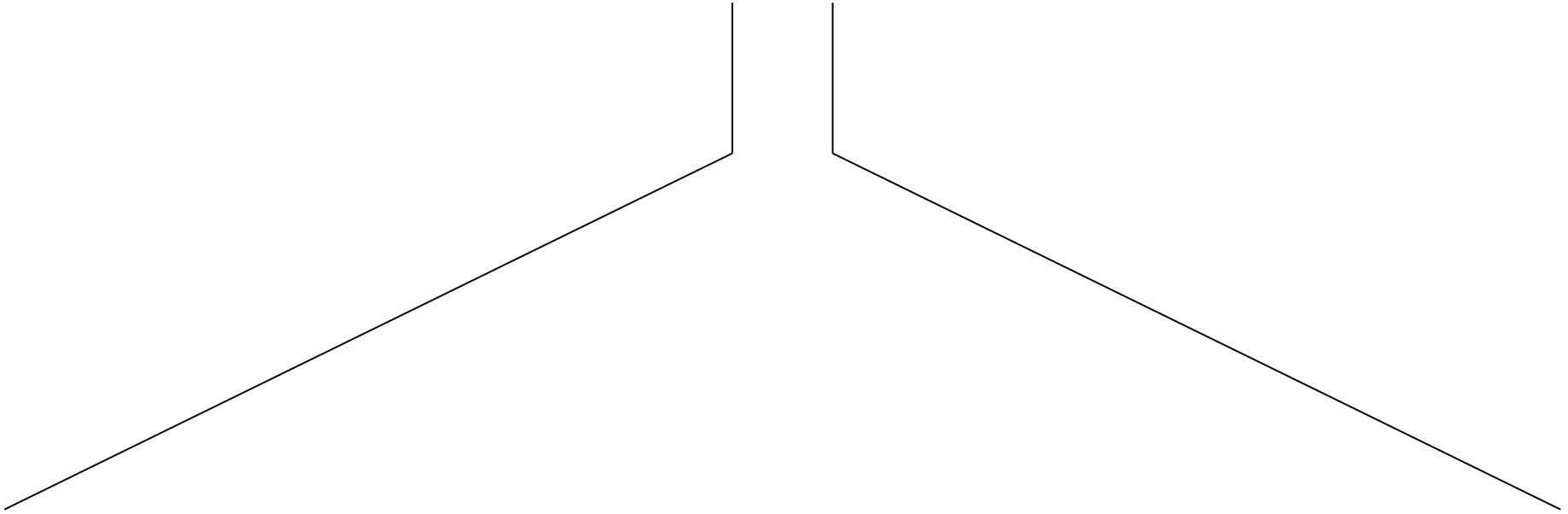
8	33
;	26
4	19
‡	16
)	16
*	13
5	12
6	11
(	10
†	8
1	8
0	6
9	5
2	5
:	4
3	4
?	3
¶	2
–	1
•	1



Implementation Strategy

The basic idea behind the program to count letter frequencies is to use an array with 26 elements to keep track of how many times each letter appears. As the program reads the text, it increments the array element that corresponds to each letter.

TWAS BRILLIG



1	1	0	0	0	0	1	0	2	0	0	2	0	0	0	0	0	1	1	1	0	0	1	0	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

To the code!