3. Arrays

Basic building blocks for programming

- any program you might want to write
  - objects
  - functions and modules
  - graphics, sound, and image I/O
  - conditionals and loops
  - Math
  - text I/O
  - primitive data types
  - assignment statements

- Ability to store and process huge amounts of data

Your first data structure

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

An **array** is an indexed sequence of values 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.
- $6.02 \times 10^{23}$ particles in a mole.

Main purpose. Facilitate storage and manipulation of data.
Processing many values of the same type

10 values, without arrays

```java
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
double x = a4 + a8;
```

tedious and error-prone code

10 values, with an array

```java
double[] a;
a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...
double x = a[4] + a[8];
```
an easy alternative

1 million values, with an array

```java
double[] a;
a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...
double x = a[234567] + a[876543];
```
scales to handle huge amounts of data

Memory representation of an array

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

A computer’s memory is also an indexed sequence of memory locations.
- Each primitive type value occupies a fixed number of locations.
- Array values are stored in contiguous locations.

for simplicity in this lecture, think of a as the memory address of the first location
the actual implementation in java is just slightly more complicated.

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, as with primitive types (stay tuned for details)

Java language support for arrays

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare an array</td>
<td>double[] a;</td>
</tr>
<tr>
<td>Create an array of a given length</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Refer to an array entry by index</td>
<td>a[i] = b[j] + c[k];</td>
</tr>
<tr>
<td>Refer to the length of an array</td>
<td>a.length;</td>
</tr>
</tbody>
</table>

Initialization options

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default initialization to 0 for numeric types</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Declare, create and initialize in one statement</td>
<td>double[] a = new double[1000];</td>
</tr>
<tr>
<td>Initialize to literal values</td>
<td>double[] x = { 0.3, 0.6, 0.1 };</td>
</tr>
</tbody>
</table>

no need to use a loop like for (int i = 0; i < 1000; ++i) a[i] = 0.0;

Important note: The code b = a does not copy an array (it makes b and a refer to the same array).

Copying an array

To copy an array, create a new array, then copy all the values.

```java
double[] b = new double[a.length];
for (int i = 0; i < a.length; i++)
    b[i] = a[i];
```

Important note: The code b = a does not copy an array (it makes b and a refer to the same array).
Programming with arrays: typical examples

Access command-line args in system array
int stake = Integer.parseInt(args[0]);
int goal = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);

Create an array with N random values
double[] a = new double[N];
for (int i = 0; i < N; i++)
a[i] = Math.random();

For brevity, N is a.length and b.length in all this code.
Copy to another array
double[] b = new double[N];
for (int i = 0; i < N; i++)
b[i] = a[i];

Print array values, one per line
for (int i = 0; i < N; i++)
System.out.println(a[i]);

Print array values, one per line
for (int i = 0; i < N; i++)
System.out.println(b[i]);

Compute the average of array values
double sum = 0.0;
for (int i = 0; i < N; i++)
sum += a[i];
double average = sum / N;

Find the maximum of array values
double max = a[0];
for (int i = 1; i < N; i++)
if (a[i] > max) max = a[i];

Copy to another array
double[] b = new double[N];
for (int i = 0; i < N; i++)
b[i] = a[i];

Pop quiz 1 on arrays

Q. What does the following code print?

```
public class PQarray1
{
    public static void main(String[] args)
    {
        int[] a = new int[6];
        int[] b = new int[a.length];
        b = a;
        for (int i = 0; i < b.length; i++)
            b[i] = 1;
        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.print(b[i] + " ");
    }
}
```

A. % java PQ1_1
0 1 2 3 4 5
0 1 2 3 4 5

Programming with arrays: typical bugs

Array index out of bounds
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
a[i] = Math.random();
No a[10] (and a[0] unused)

Uninitialized array
double[] a;
for (int i = 0; i < 9; i++)
a[i] = Math.random();
Never created the array

Undeclared variable
a = new double[10];
for (int i = 0; i < 10; i++)
a[i] = Math.random();
What type of data does a refer to?
3. Arrays

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

Example of array use: create a deck of cards

```java
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"♠", "♥", "♦", "♣"};
        String[] deck = new String[52];

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 13; j++) {
                deck[i + 13 * j] = rank[i] + suit[j];
            }
        }

        System.out.println("% java Deck");
        System.out.println(deck);
    }
}
```

Example of array use: create a deck of cards

Define three arrays
- Ranks.
- Suits.
- Full deck.

Use nested for loops to put all the cards in the deck.
Pop quiz 2 on arrays

Q. What happens if the order of the for loops in Deck is switched?

for (int j = 0; j < 4; j++)
for (int i = 0; i < 13; i++)
deck[i + 3*j] = rank[i] + suit[j];

for (int i = 0; i < 13; i++)
for (int j = 0; j < 4; j++)
deck[i + 3*j] = rank[i] + suit[j];

A. The array is filled in a different order, but the output is the same.

<table>
<thead>
<tr>
<th>j</th>
<th>suit</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

deck

NOTE: Error on page 92 in 3rd printing of text (see errata list on booksite).

Pop quiz 3 on arrays

Q. Change Deck to put the cards in rank order in the array.

% java Deck
2e 2d 2w 2s 3e 3d 3w 3s 4e 4d 4w 4s 5e 5d 5w 5s 6e 6d 6w 6s 7e 7d 7w 7s 8e 8d 8w 8s 9e 9d 9w 9s 10e 10d 10w 10s J e J d J w J s Q e Q d Q w Q s K e K d K W K s A e A d A w A s
%

% java Deck
2e 2d 2w 2s 3e 3d 3w 3s 4e 4d 4w 4s 5e 5d 5w 5s 6e 6d 6w 6s 7e 7d 7w 7s 8e 8d 8w 8s 9e 9d 9w 9s 10e 10d 10w 10s J e J d J w J s Q e Q d Q w Q s K e K d K W K s A e A d A w A s
%

A. for (int i = 0; i < 13; i++)
for (int j = 0; j < 4; j++)
deck[4*i + j] = rank[i] + suit[j];

<table>
<thead>
<tr>
<th>j</th>
<th>suit</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

deck

| deck | 2d 2w 2s 3d 3w 3s 4d 4w 4s 5d 5w 5s 6d 6w 6s 7d 7w 7s 8d 8w 8s 9d 9w 9s 10d 10w 10s 11d 11w 11s 12d 12w 12s Jd Jw Js Qd Qw Qs Kd Kw Ks Ad Aw As |
Array application: take a card, any card

**Problem:** Print a random sequence of $N$ cards.

**Algorithm**
Take $N$ from the command line and do the following $N$ times
- Calculate a random index $r$ between 0 and $1$.
- Print $\text{deck}[r]$.

**Implementation:** Add this code instead of printing $\text{deck}$ in $\text{Deck}$.

```java
for (int i = 0; i < N; i++)
    System.out.println(deck[(int)(Math.random() * 52)]);
```

**Note:** Same method effective for printing a random sequence from any data collection.

Array application: random sequence of cards

```java
public class DrawCards {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        String[] suit = {"\u2665", "\u2666", "\u2663", "\u2660"};
        String[] deck = new String[52];
        for (int i = 0; i < 13; i++)
            for (int j = 0; j < 4; j++)
                deck[i * 4 + j] = rank[i] + suit[j];

        for (int i = 0; i < N; i++)
            System.out.println(deck[(int)(Math.random() * 52)]);
    }
}
```

**Note:** Sample is with replacement (same card can appear multiple times).

Array application: shuffle and deal from a deck of cards

**Problem:** Print $N$ random cards from a deck.

**Algorithm:** Shuffle the deck, then deal.
- Consider each card index $i$ from 0 to 51.
- Calculate a random index $r$ between $i$ and 51.
- Exchange $\text{deck}[i]$ with $\text{deck}[r]$.
- Print the first $N$ cards in the deck.

**Implementation**

```java
for (int i = 0; i < 52; i++)
    { int r = i + (int)(Math.random() * (52-i));
      String t = deck[i];
      deck[i] = deck[r];
      deck[r] = t;
    }
for (int i = 0; i < N; i++)
    System.out.println(deck[i]);
```

**Note:** Each value between $i$ and 51 equally likely
Array application: shuffle a deck of 10 cards (trace)

```java
for (int i = 0; i < 10; i++)
    
    int r = i + (int)(Math.random() * (10-i));
    String t = deck[r];
    deck[r] = deck[i];
    deck[i] = t;
```

<table>
<thead>
<tr>
<th>s</th>
<th>r</th>
<th>deck</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3e</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>2e</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>6e</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>6e</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2e</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>6e</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>2e</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
<td>6e</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2e</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>6e</td>
</tr>
</tbody>
</table>

Q. Why does this method work?
- Uses only exchanges, so the deck after the shuffle has the same cards as before.
- \(N-i\) equally likely values for \(deck[i]\).
- Therefore \(N(x(N-1)\ldots x(N-i)\ldots x2) = N!\) equally likely values for \(deck[i]\).
- Initial order is immaterial.

Note: Same method is effective for randomly rearranging any type of data.

Coupon collector

**Coupon collector problem**
- \(M\) different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

**Example:** Collect all ranks in a random sequence of cards \((M=10)\)

<table>
<thead>
<tr>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>5e</td>
</tr>
</tbody>
</table>

| Collection |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | J | Q | K | A |
| 2e | 3a | 4e | 5e | 6e | 7e | 8v | 9e | 10v | Jq | Qk | Kc | Ac |
| 2v | 5e | 8a | 10p | Cq | Kc | Ac |
| 9e | 10p |

22 cards needed to complete collection

Array application: shuffle and deal from a deck of cards

```java
int N = Integer.parseInt(args[0]);
String[] suit = {"♣", "♦", "♠", "♥"};
String[] deck = new String[N];
for (int j = 0; j < N; ++j)
    deck[j] = rank[j] + suit[j];
for (int i = 0; i < N; i++)
    System.out.println(deck[i]);
System.out.println();
```

**Coupon collector simulation**
- Generate random \(int\) values between 0 and \(M-1\).
- Count number used to generate each value at least once.

**Key to the implementation**
- Create a boolean array of length \(M\).
  - Initially all false by default.
- When \(r\) generated, check the \(r\)th value in the array.
  - If \(true\), ignore it (not new).
  - If \(false\), count it as new distinct value (and set \(r\)th entry to \(true\))
Array application: coupon collector (trace for $M = 6$)

```java
boolean[] found = new boolean[M];
while (distinct < M) {
    int r = (int) (Math.random() * M);
    if (!found[r]) {
        distinct++;
        found[r] = true;
    }
}
```

Simulation, randomness, and analysis (revisited)

Once simulation is debugged, experimental evidence is easy to obtain.

Gamblers’ ruin simulation, previous lecture

```java
public class Gambler {
    public static void main(String[] args) {
        int stake = Integer.parseInt(args[0]);
        int goal = Integer.parseInt(args[1]);
        int trials = Integer.parseInt(args[2]);
        int wins = 0;
        for (int i = 0; i < trials; i++) {
            int t = stake;
            while (t > 0 && t < goal) {
                if (Math.random() < 0.5) t--;
                else t++;
            }
            if (t == goal) wins++;
        }
        System.out.println(wins + " wins out of " + trials);
    }
}
```

Analogous code for coupon collector, this lecture

```java
public class CouponCollector {
    public static void main(String[] args) {
        int M = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int cards = Integer.parseInt(args[2]);
        boolean[] found;
        for (int i = 0; i < trials; i++) {
            int distinct = 0;
            found = new boolean[M];
            while (distinct < M) {
                int r = (int) (Math.random() * M);
                if (!found[r]) {
                    distinct++;
                    found[r] = true;
                }
            }
            System.out.println(cards + " cards found after " + trials + " trials.");
        }
    }
}
```

Simulation, randomness, and analysis (revisited)

Coupon collector problem

- $M$ different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

A. (known via mathematical analysis for centuries) About $M \ln M + .57721 M$.

<table>
<thead>
<tr>
<th>Type</th>
<th>$M$</th>
<th>Expected wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>playing card suits</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>playing card ranks</td>
<td>13</td>
<td>41</td>
</tr>
<tr>
<td>baseball cards</td>
<td>1200</td>
<td>9201</td>
</tr>
<tr>
<td>Magic cards</td>
<td>12534</td>
<td>125508</td>
</tr>
</tbody>
</table>

Observed by computer simulation

<table>
<thead>
<tr>
<th>Type</th>
<th>$M$</th>
<th>Expected wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>java Coupon 4</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>java Coupon 13</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>java Coupon 1200</td>
<td>8789</td>
<td></td>
</tr>
<tr>
<td>java Coupon 12534</td>
<td>12562</td>
<td></td>
</tr>
</tbody>
</table>

Remarks

- Computer simulation can help validate mathematical analysis.
- Computer simulation can also validate software behavior.

Example: Is Math.random() simulating randomness?

Hypothesis. Centuries-old analysis is correct and Math.random() simulates randomness.
Two-dimensional arrays

A **two-dimensional array** is a *doubly-indexed* sequence of values of the same type.

### Examples

- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data
  - 

### Main purpose

Facilitate storage and manipulation of data.

---

Java language support for **two-dimensional** arrays (basic support)

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare a two-dimensional array</td>
<td><code>double[][] a;</code></td>
</tr>
<tr>
<td>Create a two-dimensional array of a given length</td>
<td><code>a = new double[1000][1000];</code></td>
</tr>
<tr>
<td>Refer to an array entry by index</td>
<td><code>a[i][j] = b[i][j] + c[i][k];</code></td>
</tr>
<tr>
<td>Refer to the number of rows</td>
<td><code>a.length;</code></td>
</tr>
<tr>
<td>Refer to the number of columns</td>
<td><code>a[1].length;</code></td>
</tr>
<tr>
<td>Refer to row ( i )</td>
<td><code>a[i]</code></td>
</tr>
<tr>
<td>Can be different for each row</td>
<td><code>a[i][0] a[i][1] a[i][2] a[i][3] a[i][4] a[i][5] a[i][6] a[i][7] a[i][8]</code></td>
</tr>
<tr>
<td>No way to refer to column ( j )</td>
<td><code>a[0][0] a[0][1] a[0][2] a[0][3] a[0][4] a[0][5] a[0][6] a[0][7]</code></td>
</tr>
</tbody>
</table>

---
Java language support for two-dimensional arrays (initialization)

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default initialization to 0 for numeric types</td>
<td>a = new double[1000][1000];</td>
<td></td>
</tr>
<tr>
<td>Declare, create and initialize in a single statement</td>
<td>double[][] a = new double[1000][1000];</td>
<td></td>
</tr>
<tr>
<td>Initialize to literal values</td>
<td>double[][] p =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ { .92, .02, .02, .02, .02 },</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ .02, .92, .32, .32, .32 },</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ .02, .02, .02, .92, .02 },</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ .92, .02, .02, .02, .02 },</td>
<td></td>
</tr>
<tr>
<td></td>
<td>{ .47, .02, .47, .02, .02 },</td>
<td></td>
</tr>
<tr>
<td></td>
<td>];</td>
<td></td>
</tr>
</tbody>
</table>

no need to use nested loops like
for (int i = 0; i < 1000; i++)
for (int j = 0; j < 1000; j++)
a[i][j] = 0.0;

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

Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

Vector addition

double[] c = new double[N];
for (int i = 0; i < N; i++)
c[i] = a[i] + b[i];

Vector dot product

double sum = 0.0;
for (int i = 0; i < N; i++)
sum += a[i]*b[i];

Mathematical abstraction: matrix
Java implementation: 2D array

Matrix multiplication

double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
c[i][j] = a[i][k] * b[k][j];

Matrix addition

double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
for (int j = 0; j < N; j++)
c[i][j] = a[i][j] + b[i][j];

Pop quiz 4 on arrays

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

1. N
2. N^2
3. N^3
4. N^4
Pop Quiz 4 on Arrays

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

- $N$
- $N^2$
- $N^3$
- $N^4$

Nested for loops: $N \times N \times N$

double[][] c = new double[N][N];
for (int $i = 0; i < N; i++$)
    for (int $j = 0; j < N; j++$)
        for (int $k = 0; k < N; k++$)
            $c[i][j] \leftarrow a[i][k] \times b[k][j]$;

Self-avoiding random walks

A dog walks around at random in a city, never revisiting any intersection.

Q. Does the dog escape?

Model: a random process in an $N$-by-$N$ lattice
- Start in the middle.
- Move to a random neighboring intersection but do not revisit any intersection.
- Outcome 1 (escape): reach edge of lattice.
- Outcome 2 (dead end): no unvisited neighbors.

Q. What are the chances of reaching a dead end?


Self-avoiding random walks

Application of 2D arrays: self-avoiding random walks

```java
public class SelfAvoidingWalker {
    public static void main(String[] args)
    {
        int $N = Integer.parseInt(args[0]);$
        int trials = Integer.parseInt(args[1]);
        int deadEnds = 0;
        for (int $t = 0; t < trials; t++)
        {
            boolean[][] a = new boolean[$N$][$N$];
            int $x = N/2, y = N/2$;
            while ($x > 0 \&\& x < N-1 \&\& y > 0 \&\& y < N-1$)
            {
                if ($a[x-1][y] \&\& a[x+1][y] \&\& a[x][y-1] \&\& a[x][y+1]$)
                    deadEnds++;
                else if ($x < 0.10$) { $a[x-1][y] = true$; }
                else if ($x > 0.90$) { $a[x-1][y] = false$; }
                $a[x][y] = true$;
            }
        }
        System.out.println(100*deadEnds/trials + "% dead ends");
    }
}
```
Simulation, randomness, and analysis (revisited again)

Self-avoiding walk in an $N$-by-$N$ lattice
- Start in the middle.
- Move to a random neighboring intersection (do not revisit any intersection).

Applications
- Model the behavior of solvents and polymers.
- Model the physics of magnetic materials.
- (many other physical phenomena)

Remark: Computer simulation is often the only effective way to study a scientific phenomenon.

Q. What is the probability of reaching a dead end?
A. Nobody knows (despite decades of study).
A. 99+% for $N > 100$ (clear from simulations).

Array: A basic building block in programming
- They enable storage of large amounts of data (values all of the same type).
- With an index, a program can instantly access a given value.
- Efficiency derives from low-level computer hardware organization (stay tuned).

Some applications in this course:

- N-body simulation
- Digital audio
- LFSR
- Digital images

Arrays: A basic building block in programming
- They enable storage of large amounts of data (values all of the same type).
- With an index, a program can instantly access a given value.
- Efficiency derives from low-level computer hardware organization (stay tuned).

Computational models play an essential role in modern scientific research.

Arrays: A basic building block in programming
- They enable storage of large amounts of data (values all of the same type).
- With an index, a program can instantly access a given value.
- Efficiency derives from low-level computer hardware organization (stay tuned).

3. Arrays

http://introcs.cs.princeton.edu