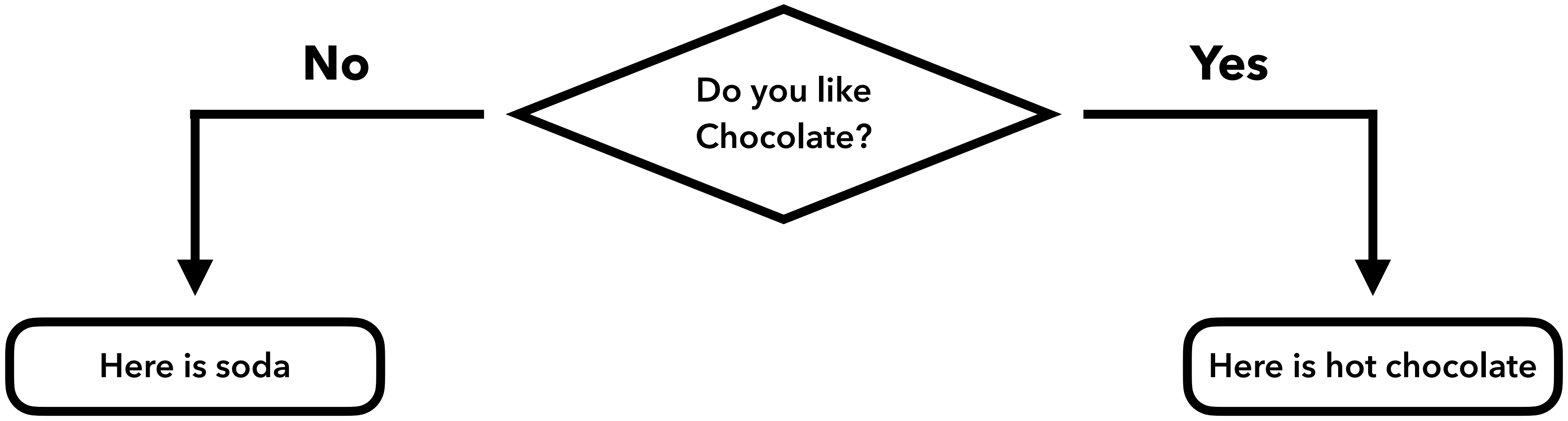
Control Flow

CS110: Introduction to Computer Science

Flow Chart


Do you like Chocolate?
No
Yes
Here is soda
Here is hot chocolate

Branching

- `if` statements allow us to optionally execute blocks of code

```
if(true-or-false-statement){
    Lines-of-code-to-run
}
```

How to make true-or-false?

- Statements that are true or false are called **boolean** statements
- The easiest way is to compare two numbers
    - **Less than** <
    - **Greater than** >
    - **Less than or equal to** <=
    - **Greater than or equal to** >=
- **Lets go through some examples**

Equal to?

- We can't use = to test equality because that's assignment.
- So, we use == instead!

```
if(x==2){
    Magic.println("Execute this");
}
```

Not Equal to?

- We can't use ≠ to test equality because it's not on the keyboard
- So, we use `!=` instead!

```
if(x!=2){
    Magic.println("Execute this");
}
```

If-Else

- Frequently, we want to execute one block of code if a statement is true and a different block if the statement is false.
- Hence, the `if-else` statement

```
if(x!=2){
    Magic.println("Execute this");
}else{
    Magic.println("Maybe this?");
}
```

if-else if-else

- We can have even more complicated conditions by using an `if-else if-else` statement

```
if(x>=2){
    Magic.println("Execute this");
}else if(x>=0){
    Magic.println("Maybe this?");
}else{
    Magic.println("When all fails");
}
```

if-else if-else if-else if-else if-else

- You can have as many `else if` statements as you want.
- Some rules:
  1. First statement must be an `if` statement.
  2. You can have **at most one** `else` statement.
  3. If you do have an `else` statement, it must be the last statement.
  4. The else if statements are evaluated **in order**.
- **Let's go through some example.**

Comparing Strings

- Java doesn't let us compare Strings directly.
- We need to use the function `equalsIgnoreCase()`.

```java
String hello = "Hello";
String next = "Goodbye";
if(hello.equalsIgnoreCase("hello")){
    Magic.println("Hello equals hello");
} else if (hello.equalsIgnoreCase(next)){
    Magic.println("Hello not equal to Goodbye");
}
```

Combining Terms

!A	A True	A False
A	False	True

- We can combine boolean expressions using three operators
  - **Not** !
  - **And** &&
  - **Or** ||
- **Let's go through examples**

A && B	B True	B False
A True	True	False
A False	False	False

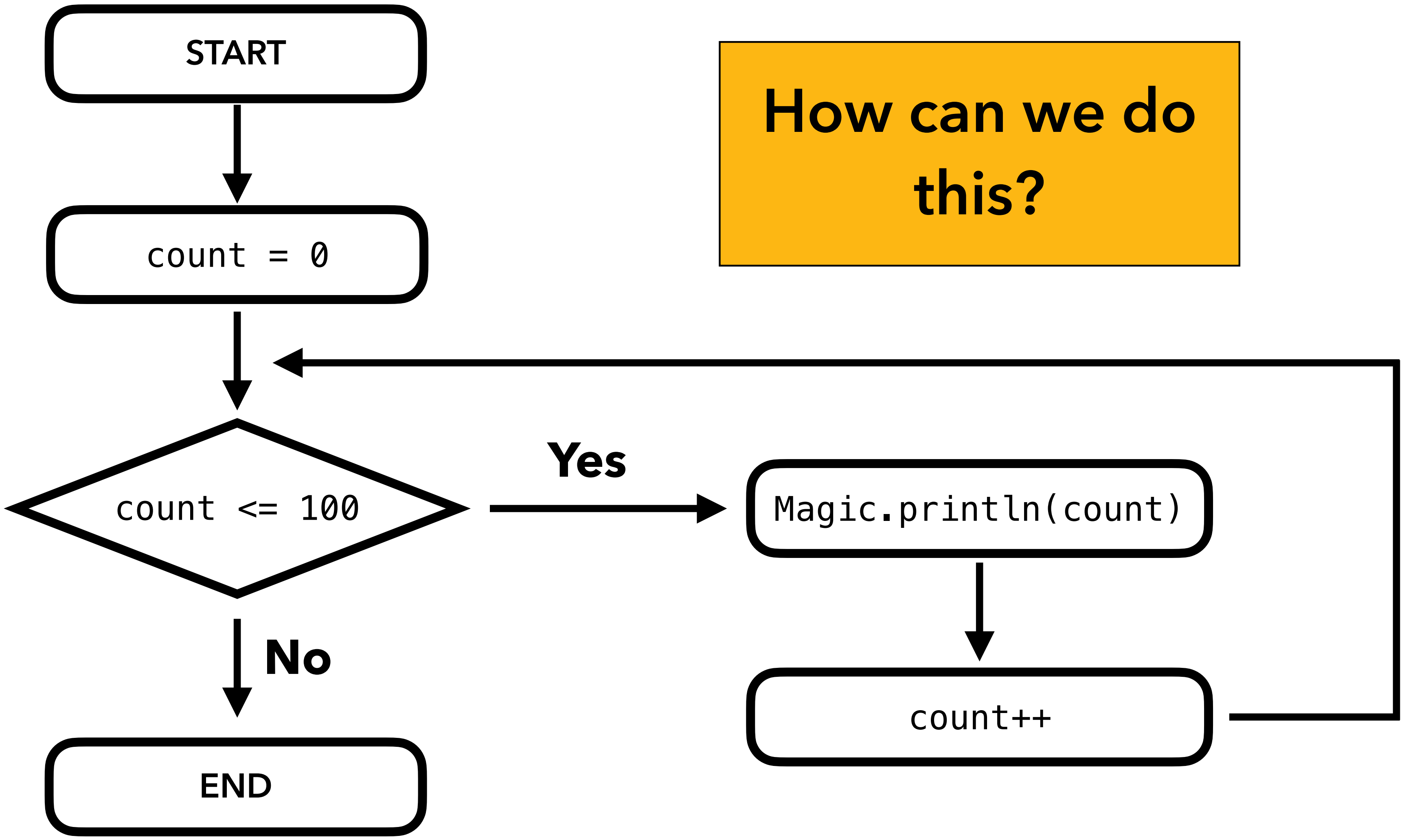
A || B	B True	B False
A True	True	True
A False	True	False

boolean

- What type of value is 2>3?
- It's a new type of variable called **boolean**.
- **boolean** variables have only two values **true** or **false**.

```
boolean a = true;
if(a){
    Magic.println("yay");
}
```

Looping



How can we do this?

Repeating Code

- We can repeat a block of code *while* something is true by using a `while` loop:

```
int count = 0;
while (count < 100){
    Magic.println("The count is "+count);
    count++;
}
```

- **Let's go through some examples.**

Common Mistakes

Forgetting the initialization

```
int count = 0;
while (count < 100){
    Magic.println("The count is "+count);
    count++;
}
```

Common Mistakes

Choosing the wrong condition

```
int count = 0;
while (count > 100){
    Magic.println("The count is "+count);
    count++;
}
```

Common Mistakes

Failure to increment

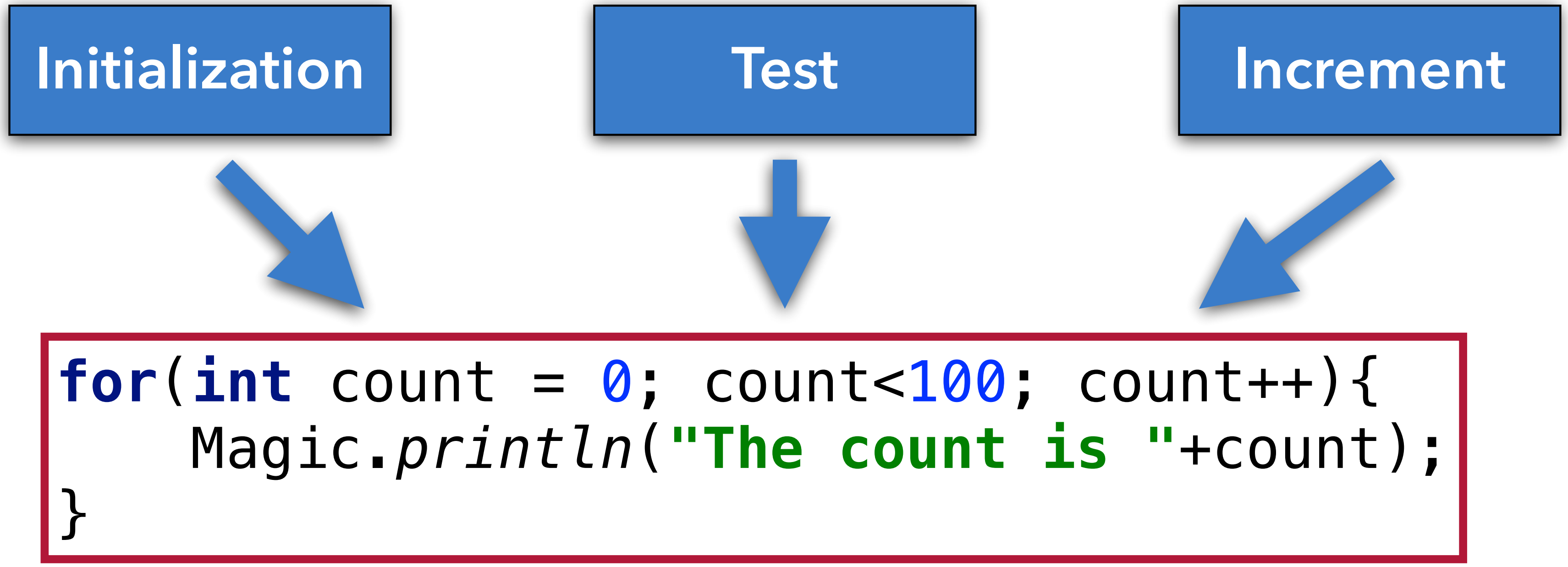
```
int count = 0;
while (count < 100){
    Magic.println("The count is "+count);
    count++;
}
```

Shortcut

- The `while` loop is flexible and it works with any boolean statement.
- However, the most common type of loop is starting at `0` and incrementing to a given value.
- The `for` loop is just such a shortcut

```java
for(int count = 0; count<100; count++){
    Magic.println("The count is "+count);
}
```

Shortcut



```
for(int count = 0; count<100; count++){
    Magic.println("The count is "+count);
}
```

`while` Loop/`for` Loop Differences

- Technically, these two loops are interchangeable.
- Generally, use the `for` loop if you are "counting" and use the `while` loop otherwise.
- There is one major subtle difference, if a variable is **declared** in the `for` loop, then you **cannot use it** after the loop is over.
- **Let's go through some examples.**

Arrays

Arrays

- Sometimes, it's helpful to keep track of a list of values.
- Java lets us do this with **arrays**.
- There are four steps to using an array:
  1. Declaration
  2. Construction
  3. Initialization
  4. Utilization

Declaration

- We declare an array by using the `[]` brackets

```
int[] myList;
```

- **Note:** Every value in the list **must be** the same type**.**

Construction

- After we declare a list, we construct it using the new operator.
- As part of constructing a list, we give it a size.

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

- The type in the construction must match the type in the declaration.
- **Note**: We can declare and construct in the same line

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

Initialization

- Like a variable, elements in the list must be given a value before they can be used.
- We reference list elements by the order they are in the list, we call this the **index**.
- The index of the first element is element zero, the index last element is size of the list minus one.

```
myList[0] = 20;
myList[1] = 200;
myList[2] = 2000;
myList[3] = 20000;
myList[4] = 200000;
```

Utilization

- We can now, use elements in the list like they are variables.

```
int total = myList[0]+myList[1]+myList[2]+myList[3]+myList[4];
```

Shortcut

- If we want to make a list and we already know the values, we can do that in one line.

```
int[] myList = {20,30,40,50,60};
```

- This is occasionally useful (less so than you might think).

Looping with Arrays

- We can use loops to go through arrays.

```
int[] myList = {20,30,40,50,60};
int total = 0;
for(int count = 0; count<myList.length; count++){
    total+=myList[count];
}
Magic.println(total);
```

We can access, the length of the array via
`listName.length`

```
int[] myList = {20,30,40,50,60};
int total = 0;
for(int count = 0; count<myList.length; count++){
    total+=myList[count];
}
Magic.println(total);
```

We can put a variable name in [ ] to reference a different index each time.

```
int[] myList = {20,30,40,50,60};
int total = 0;
for(int count = 0; count < myList.length; count++){
    total+=myList[count];
}
Magic.println(total);
```

Let's go through an example where we calculate the average

```java
int[] myList = {20,30,40,50,60};
int total = 0;
for(int count = 0; count<myList.length; count++){
    total+=myList[count];
}
Magic.println(total);
```

User Input

- Let's go through an example where we
    1. Prompt the user to input values
    2. Store those values in a list
    3. Calculate the average from the list

Shortcut for for loop

- There is an even SHORTER version of for loop that works just for arrays.
- The **for-each loop** (*aka* **enhanced for loop**)

```
int[] myList = {20,30,40,50,60};
int total = 0;

for (int element: myList) {
    total+=element;
}
```

Difference between for loops

```java
int[] myList = {20,30,40};
int total = 0;

for (int element: myList) {
    total+=element;
}
```

```java
int[] myList = {20,30,40};
int total = 0;
for(int c = 0; c<myList.length; c++){
    total+=myList[c];
}
```

In the **for-each** we can access each element directly.
In the **for loop** we need to use the index.

Which is better?

- The **for each loop** has simpler code when accessing every element, but you lose knowledge of the index.
- Also, the **for each loop** MUST go through every element. So, if you want to omit some elements, use the regular **for loop**.
- Finally, The **for loop** is more general. The **for each loop** ***only*** works with arrays.

Files and Arrays

- Magic provides a way for us to read and write integers to and from a file

```
if(!Magic.fileExists("data.txt")) {
    Magic.simpleIntFileWrite("data.txt", myList);
}
int[] dataList = Magic.simpleIntFileRead("data.txt");
for(int i =0;i<dataList.length;i++){
    Magic.println("data "+dataList[i]);
}
```

Bar Graphs

- Magic also gives us away to visualize some data via Bar Graphs!

```
int[] data = {20,30,40, 50, 60, 70};
Magic.drawGraph(data);
```