Module 2: Choice and Iteration

Ron K. Cytron

Department of Computer Science and Engineering
Washington University in Saint Louis
Thanks to Alan Waldman
for comments that improved these slides

Prepared for 2u
Semester Online

Copyright Ron K. Cytron 2013
Snappy intro monolog
2.0 Introduction

• Straight-line code
  – Statements execute in sequence
  – Program performs the same computations every time
  – Not so exciting
2.0 Introduction

• Life presents us with choices
• Programs should be able to make choices too
  – Execution becomes *conditional*
  – The arrows show the *flow* of the computation
2.0 Introduction

• Life presents us with choices
• Programs should be able to make choices too
  – Execution becomes *conditional*
  – The arrows show the *flow* of the computation
  – Sometimes our choices involve *nested* thinking

![Flowchart Image]
2.0 Introduction

- Life presents us with choices
- Programs should be able to make choices too
  - Execution becomes *conditional*
  - The arrows show the *flow* of the computation
  - Sometimes our choices involve *nested* thinking
2.0 Introduction

• In life we sometimes repeat our actions
  – Lather
  – Rinse
  – Repeat
  – Taken literally, you apparently shampoo forever

• Better to say
  – While my hair is still dirty
    • Lather
    • Rinse
    • (repeat)

• Programs should be able to iterate
2.0 Introduction

• We study here the syntax and semantics of
  – Choice
    • Choosing to do one thing, or nothing at all
    • Choosing to do one thing, or some other thing
  – Iteration
    • Repeat a computation until a desired result is achieved
      – Clean hair
    • Perform a computation for a range of values
      – Sum the integers from 1 to 10

• We see our first example of composing (nesting) program constructs
2.1 Choice Syntax and Examples

• **Usage**
  – Conditionally execute statements
  – Check for faults or problems
  – Establish dependent computations

• **Syntax**
  – “if” statement
    • With or without an “else” clause
2.1 Choice Syntax and Examples

How do we take the concept of choice and implement it in a programming language?
How do we take the concept of choice and implement it in a programming language?

The boolean data type allows us to go one way or the other.
2.1 Choice Syntax and Examples

How do we take the concept of choice and implement it in a programming language?

The boolean data type allows us to go one way or the other.

If the expression's value is true we go left.
2.1 Choice Syntax and Examples

How do we take the concept of choice and implement it in a programming language?

The boolean data type allows us to go one way or the other

If the expression's value is true, we go left
If the value is false, we go right
2.1 Choice Syntax and Examples

How do we take the concept of choice and implement it in a programming language?

The boolean data type allows us to go one way or the other.

If the expression's value is true we go left.

If the value is false, we go right.
2.1 Choice Syntax and Examples

Corresponding code

if (boolean expression) {
    statement(s) T
}
else {
    Statement(s) F
}

Flow chart
2.1 Choice Syntax and Examples

Corresponding code

```java
if (boolean expression) {
    statement(s) T
}
else {
    Statement(s) F
}
```

Flow chart

Let's look at this in terms of some real examples
2.1 Choice Syntax and Examples

Example: set the variable “max” to the value of x or y, depending on which is larger

```c
int max;

if (x > y) {
    max = x;
} else {
    max = y;
}
```
2.1 Choice Syntax and Examples

• Example: set the variable “max” to the value of x or y, depending on which is larger

• The predicate is evaluated

```c
int max;

if (x > y) {
    max = x;
} else {
    max = y;
}
```
2.1 Choice Syntax and Examples

- Example: set the variable “max” to the value of x or y, depending on which is larger

- The predicate is evaluated
  - If it is true, then max is set to the value of x
  - This happens when x is greater than y

```c
int max;
max = x;
```

```c
max = y;
```
2.1 Choice Syntax and Examples

• Example: set the variable “max” to the value of x or y, depending on which is larger

• The predicate is evaluated
  – If it is false, then max is set to the value of y

• Question: what inequality exists between x and y in this case? Answer: x <= y
2.1 Choice Syntax and Examples

• Syntax
  – The code on the left corresponds to the flow diagram shown on the right

```java
if (x > y) {
    max = x;
} else {
    max = y;
}
```
2.1 Choice Syntax and Examples

- **Evaluation**
  - Suppose we have
    - \( x = 5 \) and \( y = 7 \)

```java
if (x > y) {
    max = x;
} else {
    max = y;
}
```

![Diagram of choice syntax](image)
2.1 Choice Syntax and Examples

• Evaluation
  – Suppose we have
    • $x=5$ and $y=7$

```c
if (5 > 7) {
    max = x;
}
else {
    max = y;
}
```

![Diagram showing decision process]

```
max = x;  max = y;
```
2.1 Choice Syntax and Examples

• **Evaluation**
  – **Suppose we have**
    • \( x = 5 \) and \( y = 7 \)

```java
if (5 > 7) {
    max = x;
} else {
    max = y;
}
```

![Diagram of choice syntax](attachment:image.png)
2.1 Choice Syntax and Examples

• Evaluation
  – Suppose we have
    • \( x = 5 \) and \( y = 7 \)

```java
if (5 > 7) {
    max = x;
}
else {
    max = y;
}
```

![Diagram showing the logic of the if statement and the assignment of max to x or y based on the comparison of x and y.]
2.1 Choice Syntax and Examples

• Evaluation
  – Suppose we have
    • \( x == 5 \) and \( y == 7 \)

```java
if (5 > 7) {
    max = x;
}
else {
    max = y;
}
```

![Diagram showing the logic of the if-else statement]

```java
max = x;
max = y;
```
2.1 Choice Syntax and Examples

- **Evaluation**
  - Suppose we have
    - $x=5$ and $y=7$

```java
if (5 > 7) {
    max = x;
} else {
    max = y;
}
```

![Diagram](image)
2.1 Choice Syntax and Examples

- Boolean expression is evaluated
- If the result is "true" then statement T executes
- Otherwise statement F executes

```java
if (boolean expression) {
    statement(s) T
}
else {
    Statement(s) F
}
```
2.2 Exercise

• Video intro
  – Recall how to prompt information from a user
  – For example, how to prompt for two integers we shall call x and y
2.2 Exercise

• **Question card**
  – **Write code that**
    • Prompts the user for integer values for \(x\) and \(y\)
    • Computes the max of \(x\) and \(y\) using the “if” statement we have just studied
    • Prints the max of the two variables to the console
  – **Run your program on several different values for \(x\) and \(y\)**
    • What would constitute thorough testing of your code?
2.2 Exercise

• **Video response**
  – *Goes over an incorrect solution to this problem*
    • Sets max only one one branch, not the other
    • Declares "max" inside the if
  – *Shows that some testing may lead us to think that solution is correct*
  – *Talks about thorough testing*
    • 3 cases: < = >
  – *Goes over a correct solution to this problem*
2.3 Composition and Nesting

• An important concept in computer science is \textit{composition} – \textit{We can combine simple program components or statements to make larger ones.}

• Given the template

\begin{verbatim}
if (boolean expression) {
    Statement T
}
\end{verbatim}

\textit{We can expand Statement T to be any sequence of statements.}
2.3 Composition and Nesting

• An important concept in computer science is *composition*
  – *We can combine simple program components or statements to make larger ones*

• Given the template

```java
if (boolean expression) {
    Statement T1
}
```
2.3 Composition and Nesting

• An important concept in computer science is \textit{composition}
  
  – \textit{We can combine simple program components or statements to make larger ones}

• Given the template
  
  \begin{verbatim}
  if (boolean expression) {
    Statement T1
    Statement T2
  }
  \end{verbatim}
2.3 Composition and Nesting

• An important concept in computer science is *composition*
  – *We can combine simple program components or statements to make larger ones*

• Given the template

```java
if (boolean expression) {
    Statement T1
    Statement T2
    ...
}
```
2.3 Composition and Nesting

Example:

```java
int max;

if ( x > y ) {
    max = x;
    System.out.println("x is bigger");
}
else {
    max = y;
    System.out.println("x is not bigger");
}
```
2.3 Composition and Nesting

Example:

```java
int max;

if (x > y) {
    max = x;
    System.out.println("x is bigger");
} else {
    max = y;
    System.out.println("x is not bigger");
}
```

These statements execute in sequence if \( x > y \)
2.3 Composition and Nesting

Example:

```java
int max;

if ( x > y ) {
    max = x;
    System.out.println("x is bigger");
}
else {
    max = y;
    System.out.println("x is not bigger");
}
```

These statements execute in sequence

if x > y

The braces {.....} enclose the statements that execute when the predicate is true
2.3 Composition and Nesting

Example:

```java
int max;

if ( x > y ) {
    max = x;
    System.out.println("x is bigger");
}
else {
    max = y;
    System.out.println("x is not bigger");
}
```

These statements execute in sequence if $x \leq y$

Note again that braces serve to group these statements
2.3 Composition and Nesting

- A statement can even take the form of another “if” statement

```java
int max;

if (x > y) {
    max = x;
    System.out.println("x is bigger");
} else {
    max = y;
    if (x == y) {
        System.out.println("They are the same value");
    } else {
        System.out.println("y is bigger");
    }
}
```
2.3 Composition and Nesting

- A statement can even take the form of another “if” statement

```java
int max;

if (x > y) {
    max = x;
    System.out.println("x is bigger");
} else {
    max = y;
    if (x == y) {
        System.out.println("They are the same value");
    } else {
        System.out.println("y is bigger");
    }
}
```

Each "else" matches its closest, otherwise unmatched "if" statement
2.3 Composition and Nesting

• A statement can even take the form of another "if" statement

```java
int max;

if (x > y) {
    max = x;
    System.out.println("x is bigger");
} else {
    max = y;
    if (x == y) {
        System.out.println("They are the same value");
    } else {
        System.out.println("y is bigger");
    }
}
```

Each "else" matches its closest, otherwise unmatched "if" statement
2.3 Composition and Nesting

- A statement can even take the form of another “if” statement

```java
int max;

if (x > y) {
    max = x;
    System.out.println("x is bigger");
} else {
    max = y;
    if (x == y) {
        System.out.println("They are the same value");
    } else {
        System.out.println("y is bigger");
    }
}
```

Each "else" matches its closest, otherwise unmatched "if" statement
2.4 More examples

• The “else” part of an if statement is optional

Example

if (x < 0) {
    x = -x;
}

The above code sets x to its absolute value. If x is already positive or 0, no action is necessary. The statement inside the “if” is executed only if x is negative.
2.4 More examples

• The “else” part of an if statement is optional

Example

if (x < 0) {
    x = -x;
}

Suppose x == 5

The above code sets x to its absolute value. If x is already positive or 0, no action is necessary. The statement inside the “if” is executed only if x is negative.
2.4 More examples

• The “else” part of an if statement is optional

Example

```java
if (x < 0) {
    x = -x;
}
```

The above code sets `x` to its absolute value. If `x` is already positive or 0, no action is necessary. The statement inside the “if” is executed only if `x` is negative.

Suppose `x == 5`
2.4 More examples

• The “else” part of an if statement is optional

Example

```java
if (x < 0) {
    x = -x;
}
```

The above code sets x to its absolute value. If x is already positive or 0, no action is necessary. The statement inside the “if” is executed only if x is negative.

Suppose x == 5

The predicate is false, so the contents of the if statement are skipped.
2.4 More examples

• The “else” part of an if statement is optional

Example

```java
if (x < 0) {
    x = -x;
}
```

The above code sets `x` to its absolute value. If `x` is already positive or 0, no action is necessary. The statement inside the “if” is executed only if `x` is negative.

Suppose `x == 5` 
The program continues at  

```java
Suppose x == 5
The program continues at
```
2.4 More examples

• The “else” part of an if statement is optional

Example

```java
if (-5 < 0) {
    x = -x;
}
```

Suppose x == –5

The predicate is true, so any statements inside the braces are executed

The above code sets x to its absolute value. If x is already positive or 0, no action is necessary. The statement inside the “if” is executed only if x is negative.
2.4 More examples

• The “else” part of an if statement is optional

Example

```java
if (-5 < 0) {
    x = -x;
}
```

Suppose x == -5

The value of x at is 5

The above code sets x to its absolute value. If x is already positive or 0, no action is necessary. The statement inside the “if” is executed only if x is negative.
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x = y;
    y = t;
}
```

First, if \( x \leq y \), the predicate is false, and the statements inside do not execute. This leaves \( x \) and \( y \) alone, and they have the same values before and after the if statement.
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x = y;
    y = t;
}
```

However, if \( x > y \), then the code inside the braces executes. What does that code do? Let’s assume \( x \) is 10 and \( y \) is 5 before the if statement executes.
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x = y;
    y = t;
}
```

The value of `x` is computed and stored in a new variable named `t` (show this as columns / values)
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x = y;
    y = t;
}
```

The value of $y$ is computed and stored at $x$, destroying the value previously available there.
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x = y;
    y = t;
}
```

The value of \( t \) is computed and is stored at \( y \).

The result is that the values of \( x \) and \( y \) have been \textit{swapped}. 
2.4 More examples

• Let’s work through this example:

```java
if ( x > y ) {
    int t = x;
    x    = y;
    y    = t;
}
```

This code therefore swaps the values of x and y, if the value of x was larger then the value of y. The code sorts x and y so that after the if statement, x’s value is no larger than y’s.
Question Card

Why would the following code not have achieved the same result as our previous example?

```java
if (x > y) {
    x = y;
    y = x;
}
```
2.5a Exercise

Video Response

The code would have set x to y’s value, then set y to x’s value, which is now also the value of y. The result is that both x and y have y’s value, and the values did not swap, as they did when we used the temporary.

```java
if (x > y) {
    x = y;
    y = x;
}
```
2.5b More “if” examples

• Computer plays heads / tails
  – If random value < 0.5 print heads
    • Otherwise print tails
2.5b More “if” examples

• Computer plays heads / tails
  – If random value < 0.5 print heads
    • Otherwise print tails

• Inches to feet/inches converter
  – Reads in inches
  – Prints out f feet l inches
2.5b More “if” examples

• Computer plays heads / tails
  – If random value < 0.5 print heads
  • Otherwise print tails

• Inches to feet/inches converter
  – Reads in inches (say 67)
  – Prints out f feet i inches (say 5 feet 7 inches)
  – How you handle the special case of 1 foot or 1 inch?

• Divide two numbers carefully
  – Checks whether the divisor is 0
2.5b More “if” examples

• Computer plays heads / tails
  – If random value < 0.5 print heads
  • Otherwise print tails

• Inches to feet/inches converter
  – Reads in inches (say 67)
  – Prints out f feet i inches (say 5 feet 7 inches)
  – How you handle the special case of 1 foot or 1 inch?

• Divide two numbers carefully
  – Checks whether the divisor is 0

• Generate die throw
  – print value from 1 to 6 inclusively
2.5c More “if” examples

• Original diagrammed problem
  – Do I want to do this?
  – Is it a good idea?
  – Would it make a good story?

• Use random toss of coin to decide which to do and print the outcome and why it was chosen

• Run the program a sufficient number of times
  – So that you see each possible outcome
  – How many outcomes are there?

• What are the probabilities associated with each outcome?
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again….)
• Recall
  – Lather, rinse, repeat
2.6 Iteration

- “if” statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again....)
- Recall
  - Lather, rinse
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again....)
• Recall
  – Lather, rinse, repeat
2.6 Iteration

- “if” statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again….)
- Recall
  - Lather, rinse, repeat
  - Generates
    - Lather, Rinse, Lather, Rinse, …
2.6 Iteration

- "if" statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again….)
- Recall
  - Lather, rinse, repeat
  - Generates
    - Lather, Rinse, Lather, Rinse, …
  - We need some way to stop!
2.6 Iteration

- "if" statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again….)
- Recall
  - We need some way to stop!
  - If the hair is dirty
    - keep going
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again....)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

- “if” statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again....)
- Recall
  - We need some way to stop!
  - If the hair is dirty
    - keep going
  - Otherwise
    - stop
2.6 Iteration

- “if” statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again….)
- Recall
  - We need some way to stop!
  - If the hair is dirty
    - keep going
  - Otherwise
    - stop

hair dirty?

Lather

Rinse

yes

no
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again….)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again....)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again....)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again….)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop

hair dirty?

Lather

Rinse
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again….)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again….)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

- “if” statements allow us to choose to go one way or another
- Iteration allows us to choose to do something again (and again, and again….)
- Recall
  - We need some way to stop!
  - If the hair is dirty
    - keep going
  - Otherwise
    - stop
2.6 Iteration

• “if” statements allow us to choose to go one way or another
• Iteration allows us to choose to do something again (and again, and again....)
• Recall
  – We need some way to stop!
  – If the hair is dirty
    • keep going
  – Otherwise
    • stop
2.6 Iteration

Equivalent syntax:

while (hair dirty) {
    Lather
    Rinse
}

// here, hair is clean
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:

```
0
```
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:

- 0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:

```
0
```

Flowchart:
- **Start**
- **i = 0**
- **i < 3**
  - **Yes**: **print i**
  - **No**: **i = i + 1** (loop back)

87
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(0);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1

Diagram:

- **i = 0**
- **i < 3**
- **print i**
- **i = i + 1**
- **yes**
- **no**
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1

---

Diagram of the while loop:
- `i = 0`
- `i < 3`
- `print i`
- `i = i + 1`
- Flowchart indicating the loop condition and actions.
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1
1
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1

Diagram:

- Initial state: `i = 0`
- Condition: `i < 3`
- Loop body:
  - `System.out.println(i)`
  - `i = i + 1`
- Flow control:
  - If `i < 3` is `true`, continue the loop.
  - If `i < 3` is `false`, exit the loop.

Flowchart:

- Start at `i = 0`
- Check if `i < 3` is true
  - If true, print `i` and increment `i`
  - If false, end the loop

Sequence:

1. `i = 0` (initialization)
2. Check `i < 3`
3. If true, print `i` and set `i = i + 1`
4. Repeat from 2
5. If false, stop

Result:

- Output: `0 1`
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1

---

**Flowchart:**
- **Start**
- **i = 0**
- **i < 3**
  - **Yes**: **print i**
  - **No**: **i = i + 1**

---

99
2.6 Iteration

```java
int i = 0;
while (2 < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1

Diagram:
- Start
- i = 0
- i < 3
  - Yes: print i
  - No: i = i + 1
- End
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}

Console output:
0
1
2
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}

Console output:
0
1
2
2.6 Iteration

```java
int i = 0;
while (2 < 3) {
    System.out.println(2);
    i = i + 1;
}
```

Console output:
0
1
2
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}

Console output:
0
1
2
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1
2
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Console output:
0
1
2
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}

// Here, i >= 3

Console output:
0
1
2
2.6 Iteration

```java
while ( p ) {
    // Here, !p must be true
}
```

The loop's termination condition is always the complement of \( p \), the predicate stated in the parentheses of the `while` loop.
2.6 Iteration

```c
while ( p ) {
    // Here, !p must be true
}
```

The loop's termination condition is always the complement of \( p \), the predicate stated in the parentheses of the `while` loop.

This can be used to prove properties about programs.
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

• **Initialization**
  • Code before the while loop that establishes values necessary for the computation

• **Loop predicate**
  • Condition under which loop continues

• **Loop body**
  • Contents between the braces

• **Loop body usually contains**
  • Some statement that affects the loop's predicate
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- Initialization
  - Code before the while loop that establishes values necessary for the computation

- Loop predicate
  - Condition under which loop continues

- Loop body
  - Some statement that affects the loop's predicate
  - Contents between the braces
### 2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation
- **Loop predicate**
  - Condition under which loop continues
- **Loop body**
  - Contents between the braces
  - Some statement that affects the loop's predicate

- Loop body usually contains
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

- Loop body usually contains
  - Some statement that affects the loop's predicate
  - Otherwise the loop would never stop
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation
- **Loop predicate**
  - Condition under which loop continues
- **Loop body**
  - Contents between the braces
- **Loop termination condition**
  - The complement of the loop's predicate
  - Implied, not explicitly stated
2.6 Iteration

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

• Loop termination condition
  - The complement of the loop's predicate
  - Implied, not explicitly stated

// i >= 3
2.6b Practice using while loops

• Video Introduction
  – Randomness
  – How do we know our random number generator is fair?

• Question card: Write a program using the while loop that computes the average of 100 random numbers, each generated by Math.random()

• Response, show solution
2.6c More practice using while loops

• Video Introduction
  – Let's study the average based on how many random samples we compute
  – We want to run our previous program with the number of samples used from 1 to 1000, printing out the average each time
  – What do we expect?

• Question card:
  – What do you expect the printed averages to look like?
  – Write the program to test your hypothesis
  – What did you see?

• Response, show solution
2.6d Initial and terminal values

• Video Introduction
  – We must adapt the initialization and predicates to suit our needs

• Question card:
  – Using only the following in the loop
    • System.out.println(i) in the loop
    • i = i + something
  – Print out integers from 0 to 9 inclusively
  – From 1 to 10 inclusively
  – From 0 to 10, including 0 but excluding 10
  – From 0 to 10, including 0, excluding 10, jumping by 2 each time

• Response, show solution
2.6e Roundtable

• How many times do I have to flip a coin to get 10 heads?
• Interactive session to develop the program with a student
• Notes for RKC on next slide
How many times do I have to flip a coin to get 10 heads?

- **Concepts in this code:**
  - Number of times the coin is flipped - `numFlips`
  - Number of times heads shows up - `numHeads`

- **Initially**
  - Both are 0

- **Predicate**
  - While (`numHeads < 10`)

- **Body**
  - Flip a copy via random number
  - If heads, increment `numHeads`
  - Always increment `numFlips`

- **Now show this as code**
- **Talk about postcondition:** `numHeads >= 10`, in this case we know it is `== 10`
- **Print out numHeads**
- **Run this a few times to show the variability of outcome**

- **Computer science concept:** simulation to compute things
  - There are formulas that could tell us the answer to this question
  - But there are problems for which no formulas exist
  - So simulation is a very powerful tool to help answer such questions

- This code is in the students’ repos, and is a starting point for the upcoming material. The students should find their code and run it a few times to get a feeling for how it works. Instructions are given here to show the students how to do that.
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

Recall

- Loop body usually contains
  - Some statement that affects the loop's predicate
  - Otherwise the loop would never stop
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Recall
- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

- Loop body usually contains
  - Some statement that affects the loop's predicate
  - Otherwise the loop would never stop
2.7 The “for” loop

Recall

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

- Loop body usually contains
  - Some statement that affects the loop's predicate
  - Otherwise the loop would never stop
2.7 The “for” loop

```
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

Recall

- **Initialization**
  - Code before the while loop that establishes values necessary for the computation

- **Loop predicate**
  - Condition under which loop continues

- **Loop body**
  - Contents between the braces

- **Loop body usually contains**
  - Some statement that affects the loop's predicate
  - Otherwise the loop would never stop
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- Initialization
- Loop predicate
- Loop body
  - Some statement that affects the loop's predicate

- The while loop that simply counts has its “business” spread all over the place:
  - Initialization
  - Termination
  - Bump (increment)

- It would be nice to package that in a single programming gesture
- The “for” loop does that
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

- Initialization
- Loop predicate
- Loop body
  - Some statement that affects the loop's predicate

- The while loop that simply counts has its “business” spread all over the place:
  - Initialization
  - Termination
  - Bump (increment)

- It would be nice to package that in a single programming gesture
- The “for” loop does that
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}

for (int i=0; i < 3; i++) {
    System.out.println(i);
}
```

- Initialization
- Loop predicate
- Loop body
  - Some statement that affects the loop's predicate
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

```java
for (int i = 0; i < 3;
    System.out.println(i);
}
```

- **Initialization**
- **Loop predicate**
- **Loop body**
  - Some statement that affects the loop's predicate
2.7 The “for” loop

```java
int i = 0;
while (i < 3) {
    System.out.println(i);
    i = i + 1;
}
```

```
for (int i=0; i<3; i = i+1)
    System.out.println(i);
```
2.8 Nested Loops

• Suppose we want to run our previous problem 100 times
  – How many times do you flip a coin to get 10 heads?
• We could do this by hand, but it’s tedious
• We can use nested loops to solve this problem
  – The outer loop counts from 0 to 99
    • Use a \texttt{for} loop for this
  – The inner loop is the code we wrote before
Question Card

• Take your repo copy of the coin flipping problem
• Surround the loop there with a for loop that causes the coin flipping problem to execute 100 times.
• Run and report your results
Video Response

• Show my solution
• Have the students fix up theirs if work is needed
Video Intro

• Why run this 100 times?
  – To see on average how many flips it takes to get 10 heads

• Let’s have the user provide the number of times the simulation should run.

• New concepts:
  – Integer $N$: the number of times to run the simulation
Question Card

- Modify your code to prompt the user for N, and then run the simulation N times.
Video Response

• Show my solution
Video Response

• How many times, on average, do we have to flip a coin to see 10 heads?

• New concepts:
  – Sum of numFlips over all simulations
  – Number of simulations (N – so this isn’t really a new concept, but we need N to compute the average)
Question Card

• Modify your code to compute
  – `sumNumFlips`: the sum of the number of flips over all simulations to obtain 10 heads in each simulation

• Print the average: `sumNumFlips/N`
  – Where does this print statement appear?
Video Response

• My solution
2.9 Conclusion

• **if statements**
  – With or without the else clause
  – Allow programs to alter path of execution

• **while loops**
  – Initialize before hitting the while
  – As long as the loop predicate remains true
    • execute the code within the loop's body

• **Principle of composition**
  – Allows nesting of both constructs
  – As well as inclusion of declarations and assignment statements