

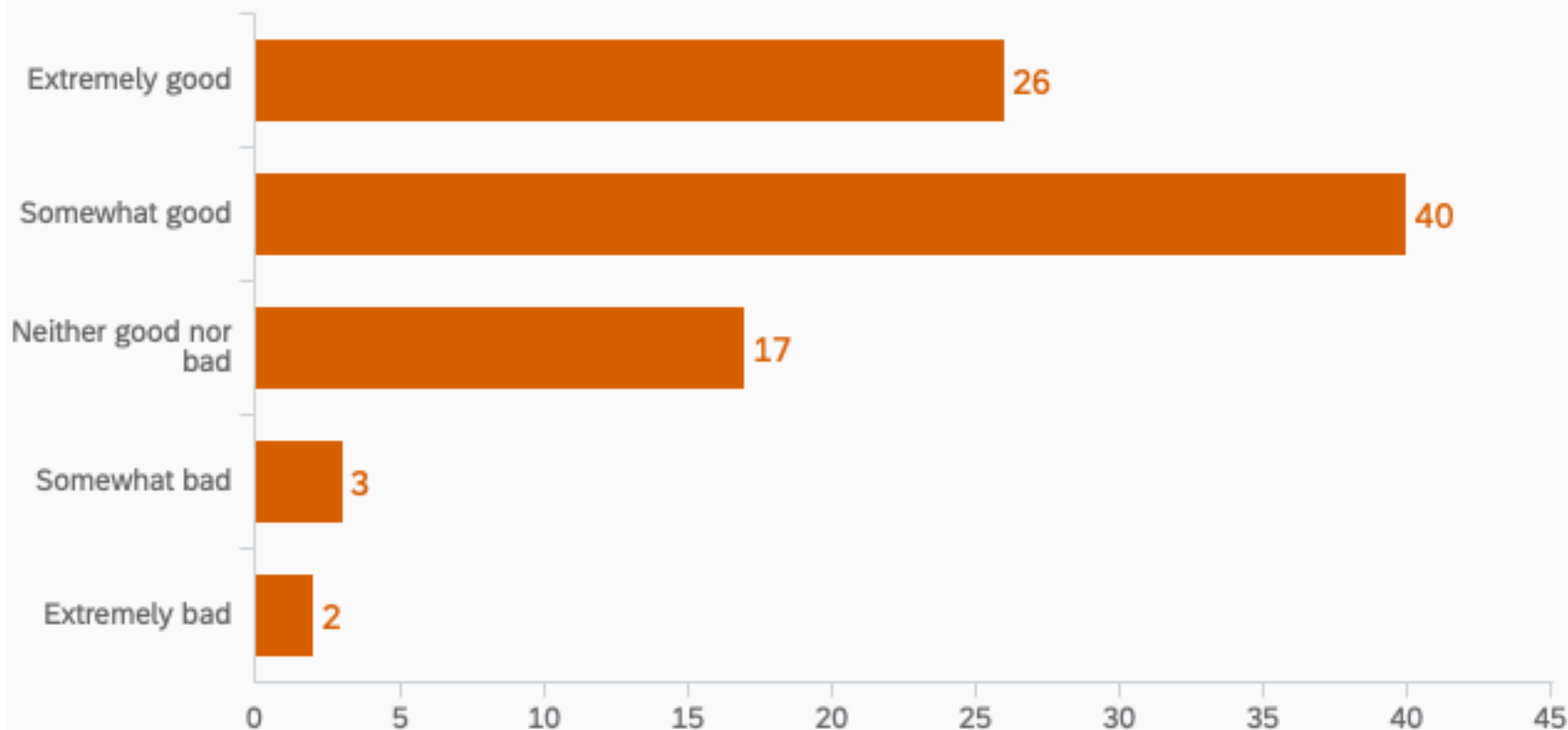
WHY I TRY NOT TO BE
PEDANTIC ABOUT CONDITIONALS.

Survey Results

- Last week we did a mid-course feedback to see if any adjustments should be made to the course
- Thank you to everyone who responded, we had 88 respondents (out of ~ 154 students)
- I'll present the results first, and then provide some commentary and modifications at the end.

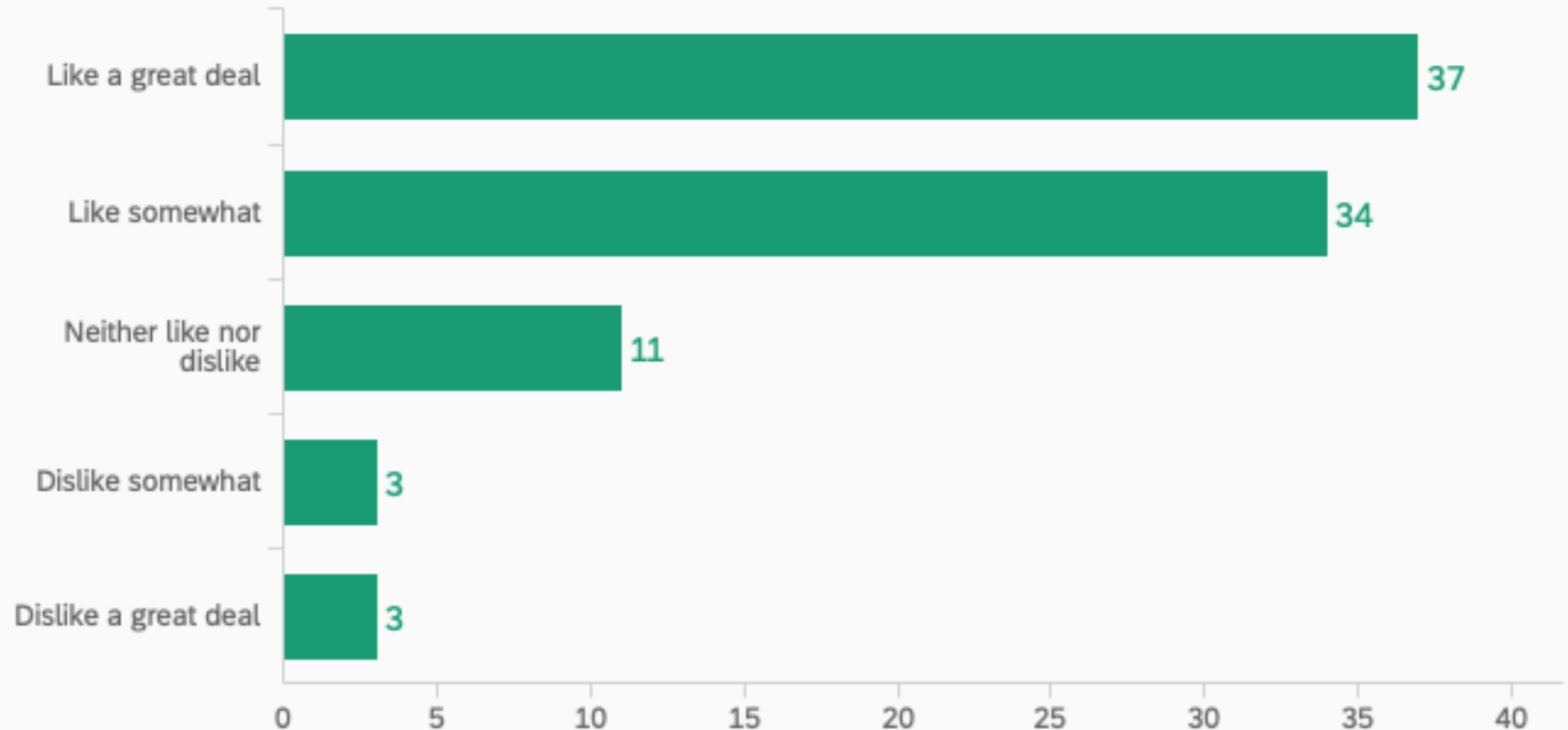
Survey Results

What do you think of the Course YouTube videos so far?



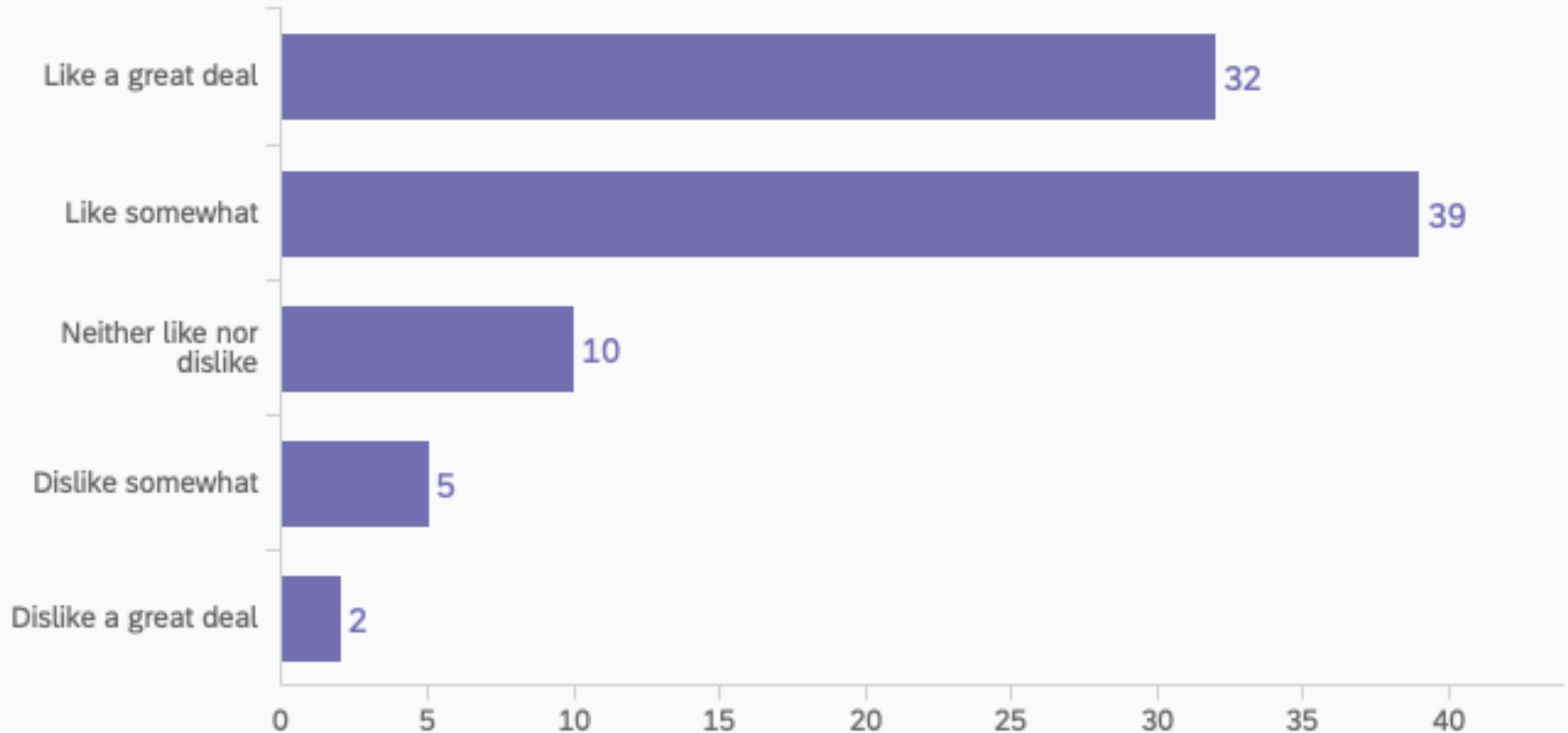
Survey Results

What do you think about the labs so far?



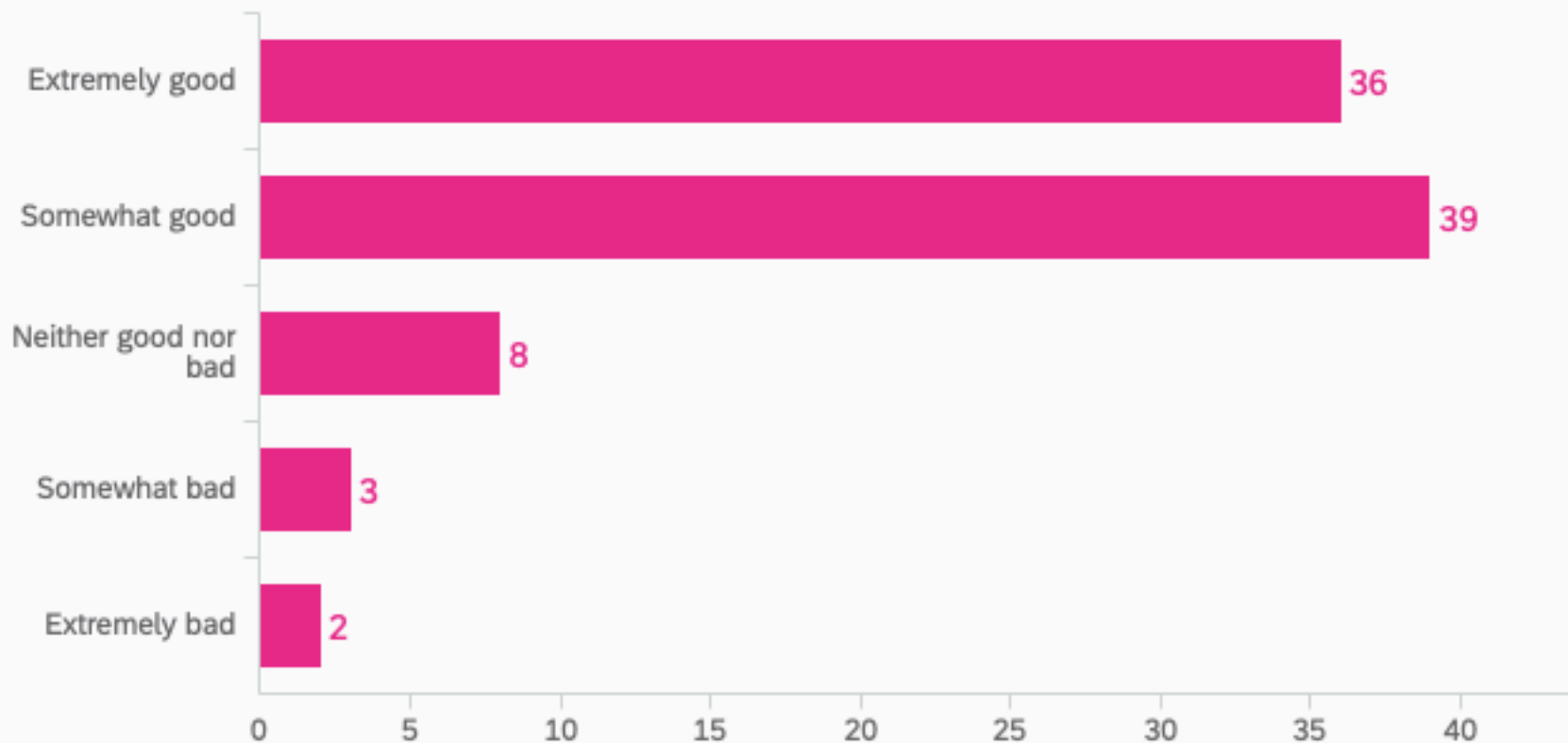
Survey Results

What do you think of the Lecture Activities so far?



Survey Results

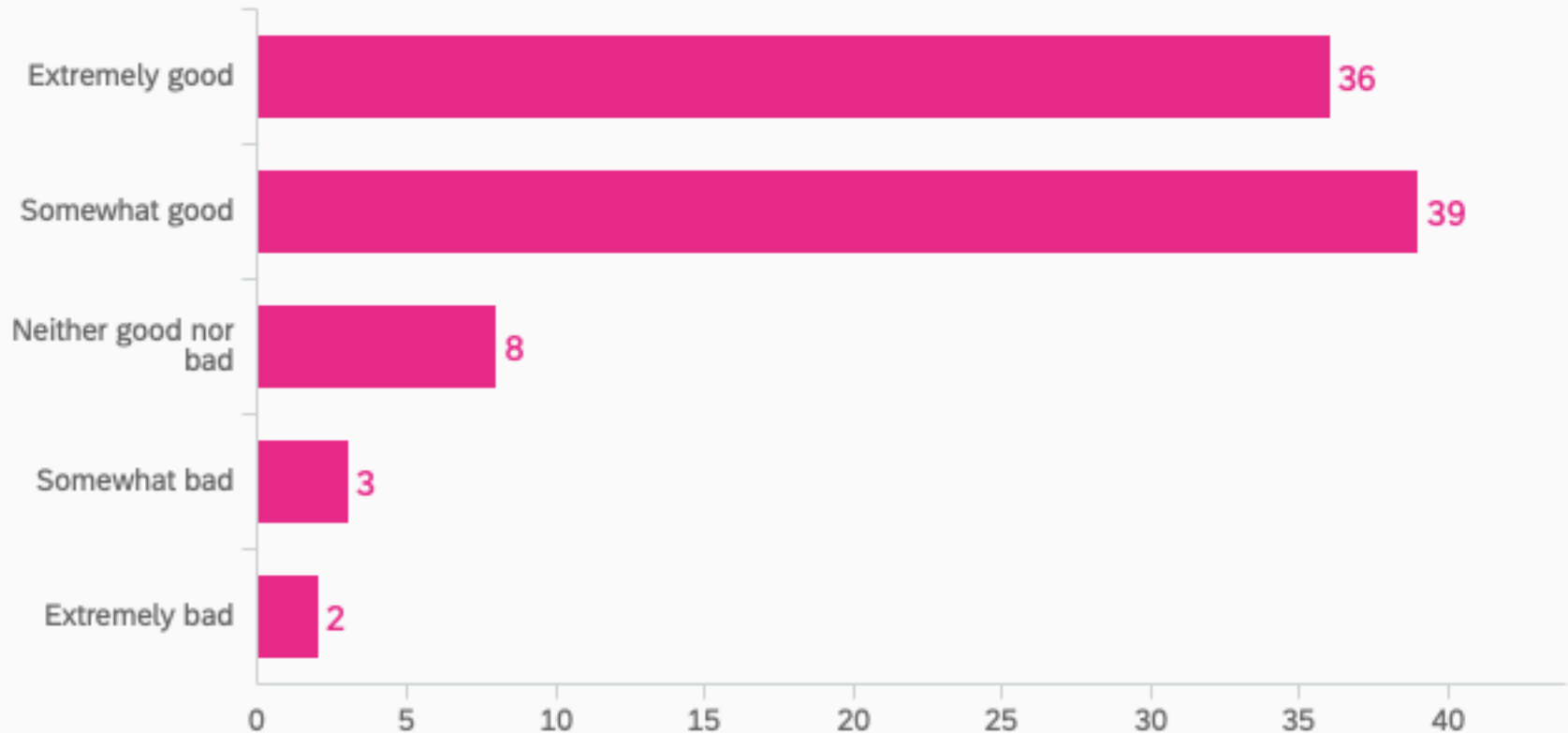
How do you think the course is going so far (overall) ?



Survey Results

How do you think the course is going so far (overall) ?

Phew! 😅



That doesn't mean things can't improve...

For all of the above comments (and more) about workload distribution, I think there are a few improvements I can suggest:

1. For the rest of the **Lecture Activities**, I will extend the due date from **Thursday 6 PM to Saturday 6 PM**. That should give you an extra couple of days of breathing room. This due-date also has the normal grace period.
2. I have adjusted the course and lecture/lab schedule so **you have more time between when you first see content and when you have to work with it**. It is unfortunate our class is on the first day of the week so I hope this change helps you all.
3. To accomplish #2, one consequence is that **if you learn content in Week X, the lab session for that content will be in Week X+1**. You will notice that as this week, Lab 6 is on Conditionals.
4. In general though, I feel that the workload for this course is fair and reasonable **each 3-credit course at UBCO should be about 6-9 hours in total**. We have gotten guidance that. If you are spending much more than that, perhaps it might be good to come and visit the TAs or myself in office hours and get some tips.



COSC 111


Computer Programming I

Loops

Dr. Firas Moosvi

Motivation

100
times



```
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
  
...  
  
...  
  
...  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");  
System.out.println("I will not throw paper airplanes in class!");
```

Tedious??

Repetition statement (loops)

Repetition statement: an action is repeated as long as a condition is true. We have two types of loops:

■ Counter-controlled repetition

- Repetition will exit after running a certain number of times.
- In a counter-controlled repetition, we have
 - a **control variable** that acts as a **counter**.
 - The control variable should be **initialized** before running the loop for the first time.
 - the control variable should then be **modified** (incremented or decremented) each time through the loop.
 - the **loop-continuation-condition** determines if looping should continue.

■ Sentinel-controlled repetition

- it is not known in advance how many times the loop will be executed.
- A sentinel value (also called a signal value) is used to decide when to exit the loop.
- The sentinel value is specially used when reading and processing a set of values from, e.g., user input, text file, array, etc.

Repetition in Java

Java provides the following mechanisms to implement loops:

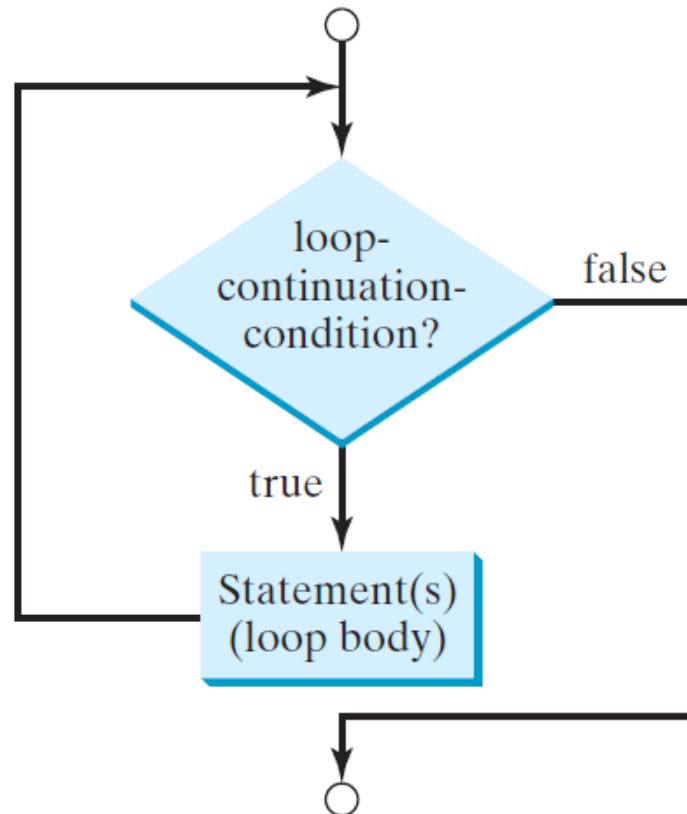
- while loops
 - also “do-while” loops

- for loops

while loops

while loop syntax

```
while (loop-continuation-condition) {  
    Statement(s); //loop body  
}
```



(A) Counter-controlled repetition

```
initialize a counter;  
while (loop-continuation-condition) {  
    Statement(s); //loop body  
    modify the counter;  
}
```

count acts as a counter

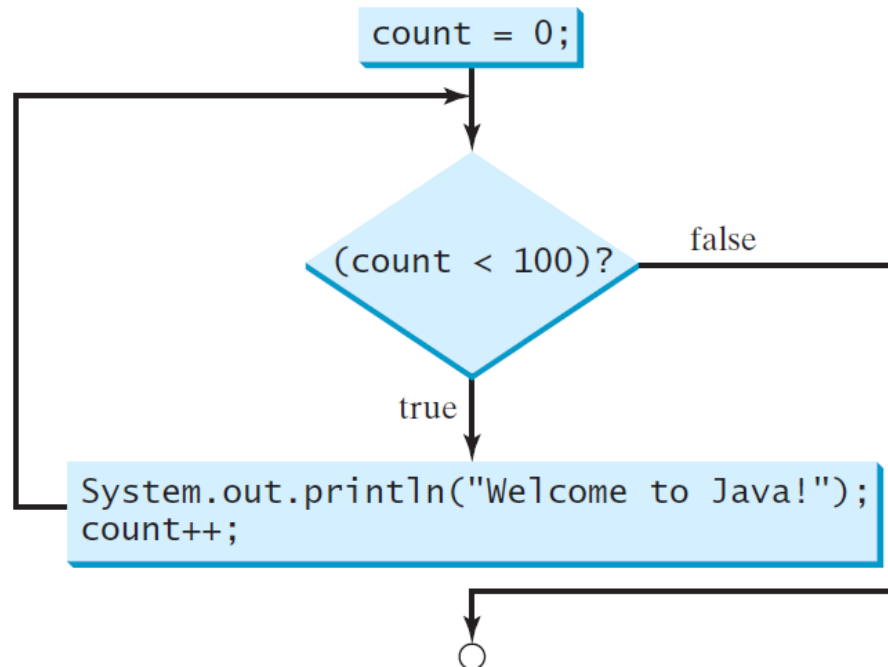
Condition is checked before
running the loop

count is incremented (to keep
record of how many times we
went through the loop

```
int count = 0;  
  
while (count < 100) {  
    System.out.println("Welcome to Java");  
    count++;  
}
```

(A) Counter-controlled repetition

```
initialize a counter;  
while (loop-continuation-condition) {  
    Statement(s); //loop body  
    modify the counter;  
}
```



Examples (A1)

Write a Java loop that prints the character '*' 10 times on the same line.

Solution: all following code segments provide the same functionality. First one is preferred.

```
int count = 0;
while (count < 10) {
    System.out.print('*');
    count++;
}
```

```
int count = 1;
while (count <= 10) {
    System.out.print('*');
    count++;
}
```

```
int count = 10;
while (count < 20) {
    System.out.print('*');
    count++;
}
```

```
int count = 10;
while (count > 0) {
    System.out.print('*');
    count--;
}
```

Examples (A2)

Write a Java loop that prints the numbers from 1 to 10.

Solution:

```
int count = 1;
while (count <= 10) {
    System.out.println(count);
    count++;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

Write a Java loop that prints the numbers from 10 to 1.

Solution:

```
int count = 10;
while (count >= 1) {
    System.out.println(count);
    count--;
}
```

Output

```
10
9
8
7
6
5
4
3
2
1
```

Examples (A3)

Write a Java loop that prints the **even** numbers from 0 to 10.

Solution:

```
int count = 0;
while (count <= 10) {
    System.out.println(count);
    count += 2;
}
```

```
int count = 0;
while (count <= 10) {
    if(count % 2 == 0)
        System.out.println(count);
    count++;
}
```

Write a Java loop that prints the **odd** numbers from 0 to 10.

Solution:

```
int count = 1;
while (count <= 10) {
    System.out.println(count);
    count += 2;
}
```

```
int count = 0;
while (count <= 10) {
    if(count % 2 != 0)
        System.out.println(count);
    count++;
}
```

Clicker Question

What is the output?

```
int i = 10;
while (i > 0)
{
    System.out.print(i) ;
    i--;
}
```

- A. nothing
- B. error
- C. 0
- D. The numbers 10,9, ...,0
- E. The numbers 10,9, ...,1

Clicker Question

What is the output?

```
int i = 0;
while (i <= 10) ;
{
    System.out.print(i) ;
    i++;
}
```

- A. infinite loop
- B. 10
- C. The numbers 0,1,2, ...,9
- D. The numbers 0,1,2, ...,10
- E. The numbers 0,1,2, ...,11

Clicker Question

What is the output?

```
int i = 0;  
while(i <= 10) {  
    System.out.print(i) ;  
}
```

- A. nothing
- B. Infinite loop
- C. 11
- D. The numbers 0,1,2,3,...,9
- E. The numbers 0,1,2,3,...,10

Practice

Write a Java loop that finds the sum and average of all numbers between 1 to 10 inclusive.

Algorithm:

- Initialize a variable, $\text{sum} = 0$
- Initialize a counter = 1
- Repeat the following 10 times:
 - Add counter to sum
 - Increment counter by 1
- Find average ($\text{avg} = \text{sum}/10$)
- Print result

Practice, cont.

Write a Java loop that finds the sum and average of all numbers between 1 to 10 inclusive.

Solution:

```
public class sum10 {  
    public static void main(String[] args) {  
        int sum = 0;  
        int count = 1;  
        while (count <= 10){  
            sum = sum + count; //or sum += count;  
            count++;  
        }  
        double avg = sum/10.0;  
        System.out.printf("Sum: %d\nAverage: %.2f\n", sum, avg);  
    }  
}
```


Practice

Write Java programs to do each of the following:

- Find the sum and average of all even numbers between 40 to 100 inclusive
- Find the sum and average of all odd numbers between 40 to 100 inclusive
- Find $10!$ (the factorial of 10) which is the product of the integers between 1 and 10.

Floating Point Accuracy in Java

Don't use floating-point values for equality checking in a loop control.

Since floating-point values are approximations for some values, using them could result in imprecise counter values and inaccurate results.

- Consider the following code for computing the result of $1 + 0.9 + 0.8 + \dots + 0.1$:

```
double item = 1;
double sum = 0;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```

(B) Sentinel-controlled repetition

```
//initialize sentinel  
while (sentinel-condition) {  
    Statement(s); //loop body  
    modify the sentinel;  
}
```

Here, the number of times a loop is executed is **not predetermined**.

You may use an input value to signify the end of the loop. Such a value is known as a sentinel value.

Examples (B1)




Write a program that reads and calculates the **sum** of an ***unspecified*** number of integers. The input 0 signifies the end of the input.

Algorithm:

- Initialize a variable, $\text{sum} = 0$
- Get the first number from the user (the sentinel)
- Until the user enters 0, repeat the following :
 - Add the user's input to sum
 - Get the next number from the user
- Print result

to get into the while loop for the first time.



Examples (B1) , solution.



Program to compute the sum of unspecified number of entries:

```
import java.util.Scanner;

public class SentinelValue {
    public static void main(String[] args) {
        // Read an initial data
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an integer (the input ends if it is 0): ");
        int data = input.nextInt();

        // Keep reading data until the input is 0
        int sum = 0;
        while (data != 0) {
            sum += data;

            // Read the next data
            System.out.print("Enter an integer (the input ends if it is 0): ");
            data = input.nextInt();
        }

        System.out.println("The sum is " + sum);
    }
}
```

Read first value
of a sentinel,
named *data*

Condition is
checked before
running the loop

read next value
of the sentinel,
data

Examples (B2)

Write a program that reads and calculates the **average** of an unspecified number of integers. The input 0 signifies the end of the input.

Algorithm:

- Initialize a variable, $\text{sum} = 0$
- Initialize a counter to zero (*to count the number of entered values*)
- Get the first number from the user (the sentinel)
- Repeat the following until the user enters 0:
 - Add the user's input to sum
 - Get the next number from the user
 - Increment counter by 1
- If the counter > 0 , calculate the average ($\text{sum} / \text{count}$)
- Print result

Examples (B2) – solution

Program to compute the average of unspecified number of entries:

```
import java.util.Scanner;

public class SentinelValue {
    public static void main(String[] args) {
        double average=0;
        // Read an initial data
        Scanner input = new Scanner(System.in);
        System.out.print("Enter an integer (the input ends if it is 0): ");
        int data = input.nextInt();
        // Keep reading data until the input is 0
        int sum = 0;
        int count = 0; //to count the number of enteries
        while (data != 0) {
            sum += data;
            count++;
            // Read the next data
            System.out.print("Enter an integer (the input ends if it is 0): ");
            data = input.nextInt();
        }
        if(count>0)
            average = (double)sum/count;
        System.out.println("The average is " + average);
    }
}
```

Examples (B3)



Suppose that the tuition for a university is \$10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

Algorithm:

- Initialize a variable tuition = 10,000
- Initialize a counter for the years, e.g. year = 0
- Repeat the following until tuition >=20,000
 - Increase tuition by 7%
 - Increment year
- Print result

```
double tuition = 10000;  
int year = 0 // Year 0  
tuition = tuition * 1.07; year++; // Year 1  
tuition = tuition * 1.07; year++; // Year 2  
tuition = tuition * 1.07; year++; // Year 3  
...
```


Examples (B3)- solution.



Suppose that the tuition for a university is \$10,000 this year and tuition increases 7% every year. In how many years will the tuition be doubled?

```
public class FutureTuition {
    public static void main(String[] args) {
        double tuition = 10000; // Year 0
        int year = 0;
        while (tuition < 20000) {
            tuition = tuition * 1.07;
            year++;
        }
        System.out.println("Tuition will be doubled in " + year + " years");
        System.out.printf("Tuition will be $%.2f in %1d years", tuition, year);
    }
}
```

Example (B4) – *Checking Input Validity* !!

Using while loops to check the validity of input data: Write a program that generates two random numbers between 0 and 10. The program should then prompts the user to enter an answer for a question on addition of the two numbers, and let the user enter a new answer until it is correct.

Algorithm:

- Generate two random numbers $n1$ and $n2$ between 0 and 10.
- Prompt the user to enter the answer for $n1 + n2$
- **Get the answer from the user (the sentinel)**
- **Check the answer, and if it is wrong repeat the following until the user enters the correct answer:**
 - Display an error message
 - Get the next answer from the user

Example (B4) – Solution



Using while loops to check the validity of input data: Write a program that generates two random numbers between 0 and 10. The program should then prompts the user to enter an answer for a question on addition of the two numbers, and let the user enter a new answer until it is correct.

```
import java.util.Scanner;

public class RepeatAdditionQuiz {
    public static void main(String[] args) {
        //Create two random numbers
        int n1 = (int) (Math.random() * 10);
        int n2 = (int) (Math.random() * 10);

        // Get first answer
        Scanner input = new Scanner(System.in);
        System.out.print("What is " + n1 + " + " + n2 + "? ");
        int answer = input.nextInt();

        while (n1 + n2 != answer) { //check the answer and get the next one if its wrong
            System.out.print("Wrong! Try again. What is " + n1 + " + " + n2 + "? ");
            answer = input.nextInt();
        }

        System.out.println("You got it!");
    }
}
```

Note how we use
while loops for
checking the validity
of a user input

Example (B5)



Write a program that randomly generates an integer between 0 and 100, inclusive. The program prompts the user to enter a number continuously until the number matches the randomly generated number.

For each user input, the program tells the user whether the input is too low or too high, so the user can choose the next input intelligently.

Example (B5) – Algorithm



Algorithm 1:

1. Generate an integer n from 1 to 100 inclusive
2. **Get the first sentinel value.**
3. Repeat the following until the user enters the correct answer:
 - Prompt the user to guess the number
 - Tell the user whether his/her guess is correct, too high, or too low.

Another technique to get into the while loop for the first time.

Algorithm 2: (same as Algorithm 1 except for step 2)

1. Generate an integer n from 1 to 100 inclusive
2. **Initialize the sentinel to a value that gets you into the loop.**
3. Repeat the following until the user enters the correct answer:
 - Prompt the user to guess the number
 - Tell the user whether his/her guess is correct, too high, or too low.

Example (B5) – Solution 1

Solution 1:

```
Scanner in = new Scanner(System.in);
int num = (int)(Math.random()*101);
System.out.print("Guess my number: ");
//get first guess
int guess = in.nextInt();
//check
while(guess != num) {
    if(num<guess)
        System.out.print("Too low. Try again: ");
    else
        System.out.print("Too high. Try again: ");
    guess = in.nextInt();
}
System.out.println("Good job!");
```

Example (B5) -- Solution 2

Solution 2:

```
Scanner in = new Scanner(System.in);
int num = (int)(Math.random()*101);
System.out.print("Guess my number: ");
//initialize guess to enter the loop
int guess = -1;
//check
while(guess != num) {
    guess = in.nextInt(); //get first guess
    if(num>guess)
        System.out.print("Too low. Try again: ");
    else if(num<guess) //don't print if guess = num
        System.out.print("Too high. Try again: ");
}
System.out.println("Good job!");
```

Practice

What is the output in the following two cases:

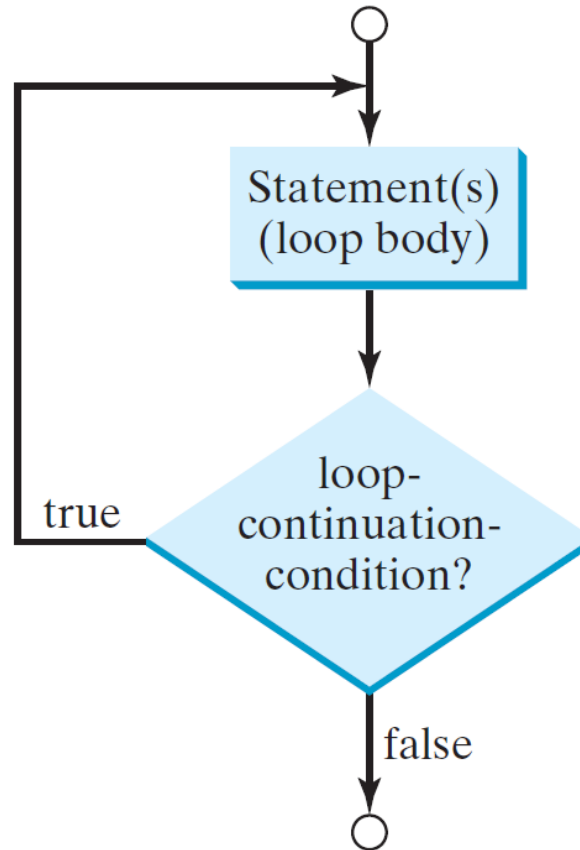
- The input is **1 2 3 4 5 0**.
- The input is **0 1 2 3 4 5 0**.

```
Scanner input = new Scanner(System.in);
int number, max;
number = input.nextInt();
max = number;
while (number != 0) {
    number = input.nextInt();
    if (number > max)
        max = number;
}
System.out.println("max is " + max);
System.out.println("number " + number);
```


do-while loops

do-while Loop

```
do {  
    Statement(s); //loop body  
} while (loop-continuation-condition);
```



A **do-while** loop is the same as a **while** loop except that it executes the loop body first and then checks the loop continuation condition.

Example (B1) Revisited



Write a program that reads and calculates the sum of an ***unspecified number of integers***. The input 0 signifies the end of the input.

Algorithm:

1. Initialize a variable, $\text{sum} = 0$
2. Do the following:
 - Get a number from the user
 - Add the number to sum
 - **If the number is not 0, Go to (2)**
3. Print result

Example , cont.



Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

```
public static void main(String[] args) {
    int data;
    int sum = 0;
    Scanner input = new Scanner(System.in);

    // Keep reading data until the input is 0
    do {
        // Read the next data
        System.out.print("Enter an integer (the input ends if it is 0): ");
        data = input.nextInt();
        sum += data;
    } while (data != 0);

    System.out.println("The sum is " + sum);
}
```

Practice

What is the output in the following two cases:

- The input is **1 2 3 4 5 0**.
- The input is **0 1 2 3 4 5 0**.

```
Scanner input = new Scanner(System.in);
int number, max;
number = input.nextInt();
max = number;
do {
    number = input.nextInt();
    if (number > max)
        max = number;
} while (number != 0);
System.out.println("max is " + max);
System.out.println("number " + number);
```

for loops

for loops

A for loop can be used to simplify writing counter-controlled while loops.

The code on the right is equivalent to the one on the left.

```
initialize a counter;  
while (loop-continuation-condition) {  
    Statement(s); //loop body  
    action-after-each-iteration;  
}
```

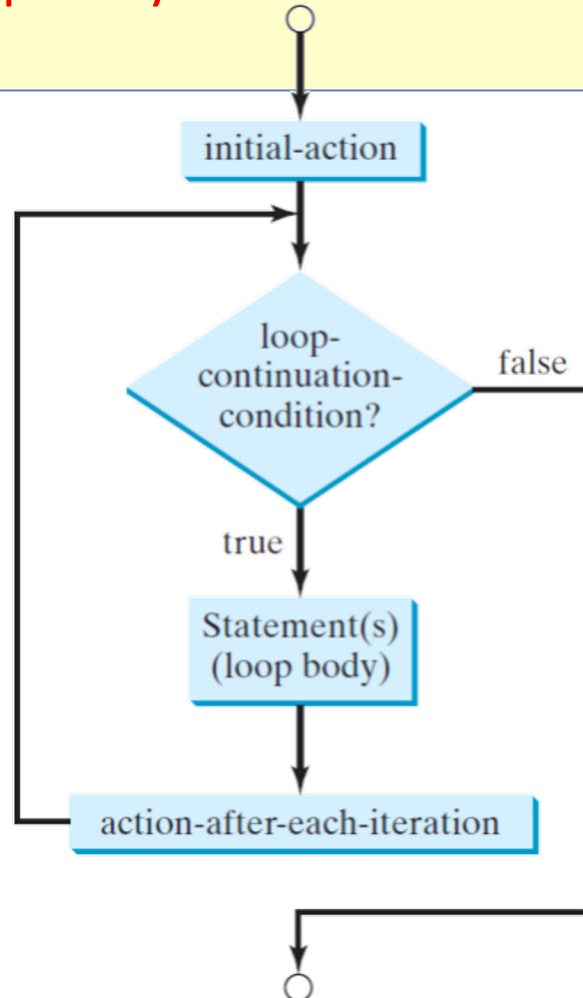
```
for (initialize counter; loop-continuation-  
    condition; action-after-each-iteration) {  
    Statement(s); // loop body  
}
```

```
int count = 0;  
  
while (count < 100) {  
    System.out.println("Welcome to Java");  
    count++;  
}
```

```
for(int counter=0; counter<100; counter++){  
    System.out.println("Welcome to Java");  
}
```

for Loop Syntax

```
for (initial-action; loop-continuation-condition; action-after-each-iteration)  
{  
    Statement(s); // loop body  
}
```



Trace for Loop

```
int i;
```

Declare i

```
for (i = 0; i < 2; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println(  
        "Welcome to Java!");  
}
```

Execute initializer
i is now 0

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println( "Welcome to Java!");  
}
```

**(i < 2) is true
since i is 0**

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```



Print Welcome to Java

System.out.println("Welcome to Java!");

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

**Execute adjustment statement
i now is 1**

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

**(i < 2) is still true
since i is 1**

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```



Print Welcome to Java

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

**Execute adjustment statement
i now is 2**

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java!");  
}
```

**(i < 2) is false
since i is 2**

Trace for Loop, cont.

```
int i;  
for (i = 0; i < 2; i++) {  
    System.out.println("Welcome to Java");  
}
```

Exit the loop. Execute the next statement after the loop

A gray callout box with a black border and rounded corners. It contains the text "Exit the loop. Execute the next statement after the loop" in bold black font. A gray arrow points from the bottom-left corner of the box to the closing curly brace of the for loop in the code block above.

Clicker Question

What is the output?

```
for (int i = 0; i <= 10; i++)  
    System.out.print(i);
```

- A. nothing
- B. error
- C. 11
- D. The numbers 0,1,2,3,...,9
- E. The numbers 0,1,2,3,...,10

Clicker Question

What is the output?

```
for (int i = 0, i <= 10, i++)  
    System.out.print(i);
```

- A. nothing
- B. error
- C. 11
- D. The numbers 0,1,2,3,...,9
- E. The numbers 0,1,2,3,...,10

Clicker Question

What is the output?

```
for (int i = 2; i < 7; i--)  
    System.out.print(i);
```

- A. nothing
- B. The loop will run many many many many times.
- C. The numbers 2,3,...,6
- D. The numbers 2,3,...,7

Note

The initial-action in a for loop can be a list of zero or more comma-separated expressions.

The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements.

Therefore, the following two for loops are correct.

```
for (int i = 1; i < 100; System.out.println(i++));
```

```
for (int i = 0, j = 0; (i + j < 10); i++, j++) {  
    // Do something  
}
```

They are rarely used in practice, however.

Note

If the loop-continuation-condition in a for loop is omitted, it is implicitly true.

- The statement given below in (a) results in an infinite loop.
- Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {  
    // Do something  
}
```

(a)


Equivalent

```
while (true) {  
    // Do something  
}
```

(b)


Caution

Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:


```
for (int i=0; i<10; i++);  Logic Error  
{  
    System.out.println("i is " + i);  
}
```


Caution, cont.

Similarly, the following loop is also wrong:

```
int i=0;
while (i < 10);  Logic Error
{
    System.out.println("i is " + i);
    i++;
}
```

In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
    System.out.println("i is " + i);
    i++;
} while (i<10);  Correct
```

Which Loop to Use?

The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms.

Use the loop statement that is most intuitive and comfortable for you. In general,

- a **for loop** may be used if the number of repetitions is known, as, for example, when you need to print a message 100 times.
- A **while loop** may be used if the number of repetitions is not known, as in the case of reading the numbers until the input is 0.
- A **do-while loop** can be used to replace a while loop if the loop body has to be executed before testing the continuation condition.

Practice

Identify and fix the errors in the following code:

```
public static void main(String[] args) {  
    for (int i = 0 , i < 10 , i++);  
        sum += i;  
    if (i < j);  
        System.out.println(i)  
    else  
        System.out.println(j);  
    while (j < 10);  
    {  
        j++;  
    }  
    do {  
        j++;  
    } while (j < 10)  
}
```

Nested Loops

Nested Loops: Example



Write a program that uses nested for loops to print a multiplication table.

```
public class MultiplicationTable {
    public static void main(String[] args) {
        // Display the table heading and number title
        System.out.println("          Multiplication Table");
        System.out.print(" ");
        for (int j = 1; j <= 9; j++)
            System.out.printf("%4d", j);
        System.out.println("\n-----");

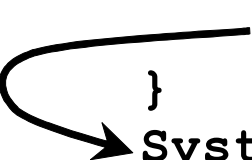
        // Print table body
        for (int i = 1; i <= 9; i++) {
            System.out.print(i + " | ");
            for (int j = 1; j <= 9; j++) {
                // Display the product and align properly
                System.out.printf("%4d", i * j);
            }
            System.out.println();
        }
    }
}
```

Keywords `break` and `continue`



Remember: we used the keyword **break** in a **switch** statement. You can also use **break** in a loop to immediately **terminate the loop**.

```
public class TestBreak {  
    public static void main(String[] args) {  
        int sum = 0;  
        int number = 0;  
  
        while (number < 20) {  
            number++;  
            sum += number;  
            if (sum >= 100)  
                break;  
        }  
        System.out.println("The number is " + number);  
        System.out.println("The sum is " + sum);  
    }  
}
```





You can also use the `continue` keyword in a loop. When it is encountered, it **ends the current iteration** and program control goes to the next iteration.

```
public class TestContinue {  
    public static void main(String[] args) {  
        int sum = 0;  
        int number = 0;  
  
        while (number < 20) {  
            number++;  
            if (number == 10 || number == 11)  
                continue;  
            sum += number;  
        }  
        System.out.println("The sum is " + sum);  
    }  
}
```


Clicker Question

How many numbers are printed?

```
for (int i=2; i < 10; i++) {  
    if (i % 2 == 0)  
        continue;  
    System.out.print(i) ;  
}
```

A. 0

B. 3

C. 4

D. 5

E. 9

Clicker Question

How many numbers are printed?

```
for (int i=2; i < 10; i++) {  
    if (i > 4)  
        break;  
    System.out.print(i) ;  
}
```

A. 0

B. 3

C. 4

D. 5

E. 9