Module 4 - Loops

General Notes

A **loop** is a program construct that repeatedly executes the loop's statements (known as the **loop body**) while the loop's expression is true; when false, execution proceeds past the loop. Each time through a loop's statements is called an **iteration**.

Sentinel Values

A **sentinel value** is a special value indicating the end of a list, such as a list of positive integers ending with `0`, as in `10 1 6 3 0`.

Increments

The statement `i = i + 1` is so common that the language supports the shorthand `++i`, with `++` known as the increment operator. (Likewise, `--` is the decrement operator, `--i` means `i = i - 1`)

There are two increment operators:

1. `++i` (pre-increment)
   - increments before evaluating to a value.
   - *Ex: If `i` is `5`, outputting `++i` outputs `6`.*
2. `i++` (post-increment)

- increments after evaluating to a value.
- *Ex: If `i` is `5`, outputting `i++` outputs `5` (and then `i` becomes `6`).*

This material avoids the in-loop
declaration approach. The authors hope to make the learning less error-prone, and have confidence that programmers can easily pick up on the common in-loop declaration approach later.

While Loop

A **while loop** is a program construct that repeatedly executes a list of sub-statements (known as the **loop body**) while the loop's expression evaluates to true.

- Each execution of the loop body is called an **iteration**.

Example 1

```
import java.util.Scanner;

public class CountUp {
  public static void main(String[] args) {
     Scanner scnr = new Scanner(System.in);
     int currPower;
     char userChar;

     currPower = 2;
     userChar = 'y';

     while (userChar == 'y') {
        System.out.println(currPower);
        currPower = currPower * 2;
        userChar = scnr.next().charAt(0);
     }

     System.out.println("Done");
  }
```

Example 2

```java
import java.util.Scanner;

public class ConvertCtoF {
   public static void main(String [] args) {
      Scanner scnr = new Scanner(System.in);
      double celsiusValue;
      double fahrenheitValue;
      char userChar;

      celsiusValue = 0.0;
      userChar = 'y';

      while (userChar == 'y') {
         fahrenheitValue = (celsiusValue * 9.0 / 5.0) + 32.0;

         System.out.print(celsiusValue + " C is ");
         System.out.println(fahrenheitValue + " F");

         System.out.print("Type y to continue, any other to quit: ");
         userChar = scnr.next().charAt(0);

         celsiusValue = celsiusValue + 5;
         System.out.println("");
      }

      System.out.println("Goodbye.");
   }
}
```

Example 3 - GCD

```java
import java.util.Scanner;

// Output GCD of user-input numA and numB

public class GCDCalc {
   public static void main(String[] args) {
      Scanner scnr = new Scanner(System.in);
      int numA;  // User input
      int numB;  // User input

      System.out.print("Enter first positive integer: ");
      numA = scnr.nextInt();

      System.out.print("Enter second positive integer: ");
      numB = scnr.nextInt();

      while (numA != numB) { // Euclid's algorithm
         if (numB > numA) {
            numB = numB - numA;
         }
         else {
            numA = numA - numB;
         }
      }

      System.out.println("GCD is: " + numA);
   }
}
```

Common Errors

- *A common error is to use the opposite loop expression than desired, like using `x == 0` rather than `x != 0`.*
- An **infinite loop** is a loop that never stops iterating. A common error is to accidentally create an infinite loop, often by forgetting to update a variable in the body, or by creating a loop expression whose evaluation to false isn't always reachable.

For Loops

A **for loop** is a loop with three parts at the top:

- A loop variable initialization
- A loop expression
- A loop variable update.

A **for loop** describes iterating a specific number of times more naturally than a while loop.

```
for (initialExpression; conditionExpression; updateExpression) {
  // Loop body
}
// Statements after the loop
```

Example 1

```java
import java.util.Scanner;

public class SavingsInterestCalc {
  public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    double initialSavings;  // User-entered initial savings
    double interestRate;    // Interest rate
    double currSavings;     // Current savings with interest
    int i;                  // Loop variable

    System.out.print("Enter initial savings: ");
    initialSavings = scnr.nextDouble();

    System.out.print("Enter interest rate: ");
    interestRate = scnr.nextDouble();

    System.out.println("\nAnnual savings for 10 years: ");

    currSavings = initialSavings;
    for (i = 0; i < 10; ++i) {
      System.out.println("$" + currSavings);
      currSavings = currSavings + (currSavings * interestRate);
    }
  }
```

Example 2

```
// Outputs 10 15 20 25 30 35 40 45 50

public class MultiplesOfFive {
  public static void main(String [] args) {
    int i;

    for (i = 10; i <= 50; i = i + 5) {
      System.out.print(i + " ");
    }

    System.out.println("");
  }
}
```

Choosing Between `for` and `while` Loops

Loop Type	Description
`for`	Number of iterations is computable before the loop, like iterating N times.
while	Number of iterations is not (easily) computable before the loop, like iterating until the input is 'q'.

Loop Style Issues

Starting With 0

Programmers in **C**, **C++**, **Java**, and other languages have generally standardized on looping `N` times by starting with `i = 0` and checking for `i < N`, rather than by using `i = 1` and `i <= N`.

- One reason is due to other constructs (arrays / vectors), often used with loops, start with `0`. Another is simply that a choice was made.

The ++ Operators

Some consider `++i` safer for beginners in case they type `i = ++i`, which typically works as expected (whereas `i = i++` does not), so this material uses `++i` throughout.

- The `--` operator also has prefix and postfix versions.
- Incidentally, the **C++** programming language gets its name from the `++` operator, suggesting **C++** is an increment or improvement over its C language predecessor.

In-loop declaration of `i`

Variables can be declared throughout code, so many programmers use: `for (int i = 0; i < N; ++i)`.

- **Remember not to declare variables within loops, re-declaring variables repeatedly.**

Common Errors / Good Practice

- A common error is to also have a `++i;` statement in the loop body, causing the loop variable to be updated twice per iteration.
- While the initialization and update parts of a `for` loop can include multiple statements separated by a comma, good practice is to use a single statement for each part.
- Good practice also is to use a `for` loop's parts to count the necessary loop iterations, with nothing added or omitted.

AVOID THESE LOOP VARIATIONS

```
// initialExpression not related to counting iterations; move r = rand() before loop
for (i = 0, r = rand(); i < 5; ++i) {
  // Loop body
}

// updateExpression not related to counting iterations; move r = r + 2 into loop body
for (i = 0; i < 5; ++i, r = r + 2) {
  // Loop body
}
```

Do-while Loops

A **do-while loop** is a loop construct that first executes the loop body's statements, then checks the loop condition.

Example 1

```java
import java.util.Scanner;

public class DoWhile {
  public static void main(String[] args) {
     Scanner scnr = new Scanner(System.in);
     String fill;

     fill = "*";

     do {
        System.out.println(fill + fill + fill);
        System.out.println(fill + fill + fill);
        System.out.println(fill + fill + fill);

        System.out.print("Enter char (q to quit): ");
        fill = scnr.next();
        System.out.println("");
     } while (!fill.equals("q"));
  }
}
```