Lecture 8:The For Loop

AP Computer Science Principles

Increment and decrement

shortcuts to increase or decrease a variable's value by 1

Shorthand	Equivalent longer version
variable++;	**variable** = **variable** + 1;
variable--;	**variable** = **variable** - 1;

```
int x = 2;
x++;                    // x = x + 1;
                        // x now stores 3

float gpa = 2.5;
gpa--;                  // gpa = gpa - 1;
                        // gpa now stores 1.5
```

Modify-and-assign

shortcuts to modify a variable's value

Shorthand	Equivalent longer version
variable += value;	**variable = variable + value;**
variable -= value;	**variable = variable - value;**
variable *= value;	**variable = variable * value;**
variable /= value;	**variable = variable / value;**
variable %= value;	**variable = variable % value;**

```
x += 3;                    // x = x + 3;

gpa -= 0.5;                // gpa = gpa - 0.5;

number *= 2;               // number = number * 2;
```

The `for` loop

Repetition with `for` loops

- So far, repeating a statement is redundant:

```
println("Homer says:");
println("I am so smart");
println("I am so smart");
println("I am so smart");
println("I am so smart");
println("S-M-R-T... I mean S-M-A-R-T");
```

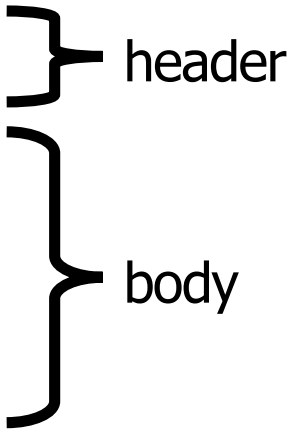
- Java's **`for` loop** statement performs a task many times.

```
println("Homer says:");

for (int i = 1; i <= 4; i++) {    // repeat 4 times
    println("I am so smart");
}
println("S-M-R-T... I mean S-M-A-R-T");
```

`for` loop syntax

```
for (initialization; test; update) {
    statement;
    statement;
    ...
    statement;
}
```



- Perform **initialization** once.
- Repeat the following:
  - Check if the **test** is true. If not, stop.
  - Execute the **statement**s.
  - Perform the **update**.

Initialization

```
for (int i = 1; i <= 6; i++) {
    println("I am so smart");
}
```

- Tells Java what variable to use in the loop
  - Performed once as the loop begins
  - The variable is called a *loop counter*
    - can use any name, not just `i`
    - can start at any value, not just `1`

Test

```
for (int i = 1; i <= 6; i++) {
    println("I am so smart");
}
```

- Tests the loop counter variable against a limit
  - Uses comparison operators:

<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

Update

```
for (int i = 1; i <= 6; i++) {
    println("I am so smart");
}
```

- The update is done at the end of each iteration of the loop.
- Can update by adding/subtract any amount. For example, i--, i+=3, i-=5 etc…

Example

- Write a for loop to print: 5 9 13 17 21 25

```
for (int i = 5; i <= 25; i+=4){
        print(i + " ");
  }
```

Repetition over a range

```
println("1 squared = " + 1 * 1);
println("2 squared = " + 2 * 2);
println("3 squared = " + 3 * 3);
println("4 squared = " + 4 * 4);
println("5 squared = " + 5 * 5);
println("6 squared = " + 6 * 6);
```

  - Intuition: "I want to print a line for each number from 1 to 6"

- The `for` loop does exactly that!

```
for (int i = 1; i <= 6; i++) {
    println(i + " squared = " + (i * i));
}
```

  - "For each integer **i** from 1 through 6, print ..."

For Loop in Movies

Groundhog Day(1993); Bill Murray.

Looper(2010); Bruce Willis and Joseph Gordon-Levitt, Emily Blunt.

Edge of Tomorrow(2014); Tom Cruise, Emily Blunt.

isPrime

```
boolean isPrime(int n) {
    int factors = 0;
    for (int i = 1; i <= n; i++) {
        if (n % i == 0) {
            factors++;
        }
    }
    if (factors == 2) {
        return true;
    }
    else {
        return false;
    }
}
```

- Calls to methods returning `boolean` can be used as tests:

```
if (isPrime(57)) {
    ...
}
```

print vs println

- The `print` comand prints without moving to a new line; the cursor stays on the same line after printing.
  - allows you to print partial messages on the same line

```
int highestTemp = 5;
for (int i = -3; i <= highestTemp / 2; i++) {
    print((i * 1.8 + 32) + "  ");
}
```

- Output:

```
26.6  28.4  30.2  32.0  33.8  35.6
```

  - Concatenate `"  "` to separate the numbers

Counting down

- The **update** can use `--` to make the loop count down.
  - The **test** must say `>` instead of `<`

```
print("T-minus ");
for (int i = 10; i >= 1; i--) {
     print(i + ", ");
}
println("blastoff!");
println("The end.");
```

  - Output:

```
T-minus 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, blastoff!
The end.
```

Scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the `{ }` braces
    - A variable declared in a `if` or `for` block exists only in that block.
    - A variable declared in a method exists only in that method.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                    // ERROR: overlapping scope
    print(" ");
}
i = 4;                            // ERROR: outside scope
```

Scope implications

```
for (int i = 1; i <= 100; i++) {
    print(“A");
}
for (int i = 1; i <= 100; i++) {    // OK
    print(”BB");
}
int i = 5;   // OK: outside of loop's scope
```

Cumulative algorithms

Adding many numbers

- How would you find the sum of all integers from 1-1000?

```
// This may require a lot of typing
int sum = 1 + 2 + 3 + 4 + ... ;
println("The sum is " + sum);
```

- What if we want the sum from 1 - 1,000,000?
  Or the sum up to any maximum?
  - How can we generalize the above code?

Cumulative sum loop

```
int sum = 0;
for (int i = 1; i <= 1000; i++) {
    sum = sum + i; // or sum += i;
}
println("The sum is " + sum);
```

- **cumulative sum**: A variable that keeps a sum in progress and is updated repeatedly until summing is finished.
  - The `sum` in the above code is an attempt at a cumulative sum.
  - Cumulative sum variables must be declared *outside* the loops that update them, so that they will still exist after the loop.

Cumulative product

- This cumulative idea can be used with other operators:

```
int product = 1;
for (int i = 1; i <= 20; i++) {
    product = product * 2;
}
println("2 ^ 20 = " + product);
```

Keep For Loops Simple!

- Keep for loops's tests and update as simple when possible. For example, i begins at 0 or 1 and goes up to n and updating by 1: i++.
- If more complicated index processing is necessary, use if conditional.
- What does this do?

```
for (int i = 1; i <= 20; i++) {
  if(i % 2 == 0){
   print(i + " ");
  }
}
// prints only even integers 1 to 20
```

In Class Practice

- Write a for loop that prints all the numbers from 1 to 100, separated by spaces all on the same line.

- Prints all the even numbers from 1 to 1000 separated by spaces. Your loop counter needs to start at 1 and go up to 1000 and increment by 1 and use a conditional.

- Prints all multiples of 3 but not of 4 from 1 to 1000 separated by spaces. Your loop counter needs to start at 1 and go up to 1000 and increment by 1.

Lab 1

Go to my codingbat homepage at:

https://codingbat.com/home/lnguyen8@bostonpublicschools.org

And do all the problems from the ”for” page.

Remember to login to save your work!

Lab 2

In Processing, create a window size whose width and height are both multiples of a 100 (for example, size(500,700)).

Use two for loops: one to draw horizontal lines and one for vertical lines such that the window divided evenly into a grid of boxes each with dimension 100 pixels x 100 pixels. Both loops must use the reserved variables width and height so that your program will still work if it is resized.

One for loop must increment by 1, i.e. i++ while the other for loop must increment by 100, i.e. i+=100.

Use line(x1,y1,x2,y2). Use strokeWeight(int) for line thickness and stroke(r,g,b) for line color.

Homework

1) Read and reread these lecture notes.

2) Complete the problem set on For Loops.

3) Complete the labs.

References

Part of this lecture is taken from the following book.

Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.