Programming 1

Lecture 4 – Loops & Arrays

Faculty of Information Technology
Hanoi University

Contents

- `for` loop
- Arrays
- Stopping a `for` loop with `break`
- Skipping `for` loop with `continue`
- `while` loop & `do..while` Loop

The power of computer

- Comes from the fact that it can repeat things efficiently.
  - They are extremely fast!
- Statements can be made to repeat for a great number of times.
  - Most problems require repeating a lot of calculations or actions.
- All programming languages support repetition with a feature called **Loop**.

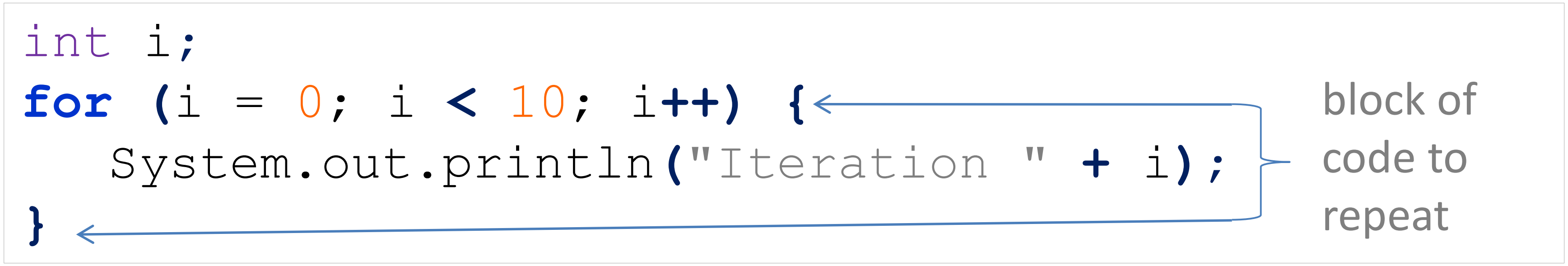
Motivation problem

- ❓ Suppose that you want to print all integers from 1 to 1000.
  - ❖ Can you write `System.out.println` 1000 times?

- ➢ **Solution:** let computers perform what they are good at: repetitive work by using loop

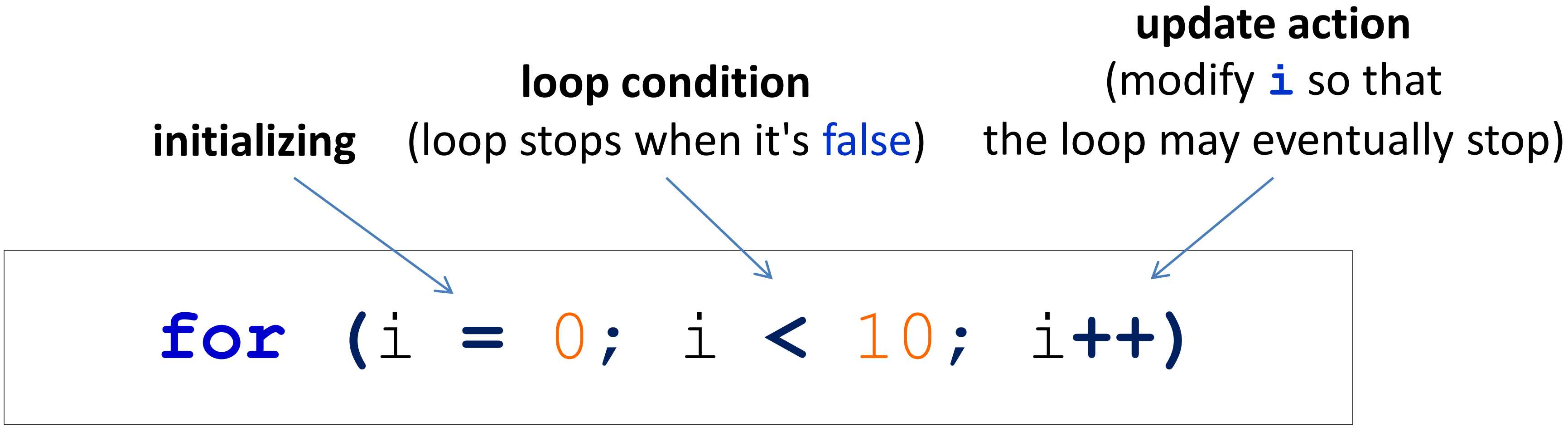
The for loop

- Used to repeat a block of code many times.



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

block of code to repeat



initializing

loop condition (loop stops when it's false)

update action (modify i so that the loop may eventually stop)

```
for (i = 0; i < 10; i++)
```

The for loop explained

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

- Let `i = 0`
- Now `i < 2` is **`true`**, let's display the text:

  `Iteration 0`
- Execute **`i++`**, and now **`i`** becomes **`1`**
- The condition **`i < 2`** is still **`true`**, let's display the text:

  `Iteration 1`
- Execute **`i++`**, and now **`i`** becomes **`2`**
- Finally **`i < 2`** is **`false`**, we won't display another line.Loop ends.

  How many lines have we displayed? What numbers were shown?

Example

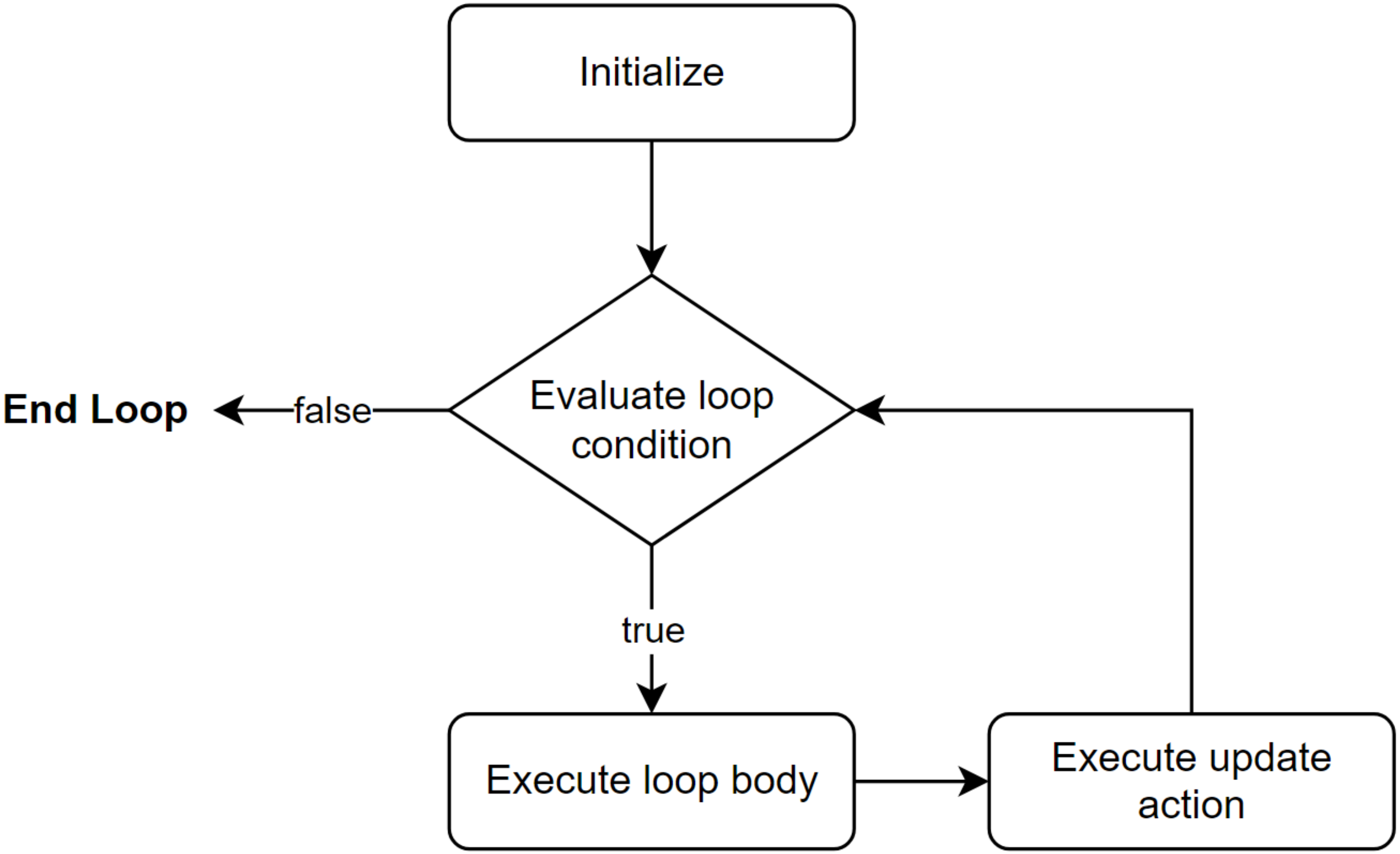
```java
System.out.println("The first 10 natural numbers:");
for (int i = 1; i <= 10; i++) {
   System.out.println(i);
}
```

Result

```
The first 10 natural numbers:
1
2
3
4
5
6
7
8
9
10
```

- Just change number 10 in to 1000 to print the first 1000 natural numbers.

`for` loop flowchart



Initialize

Evaluate loop condition

false

End Loop

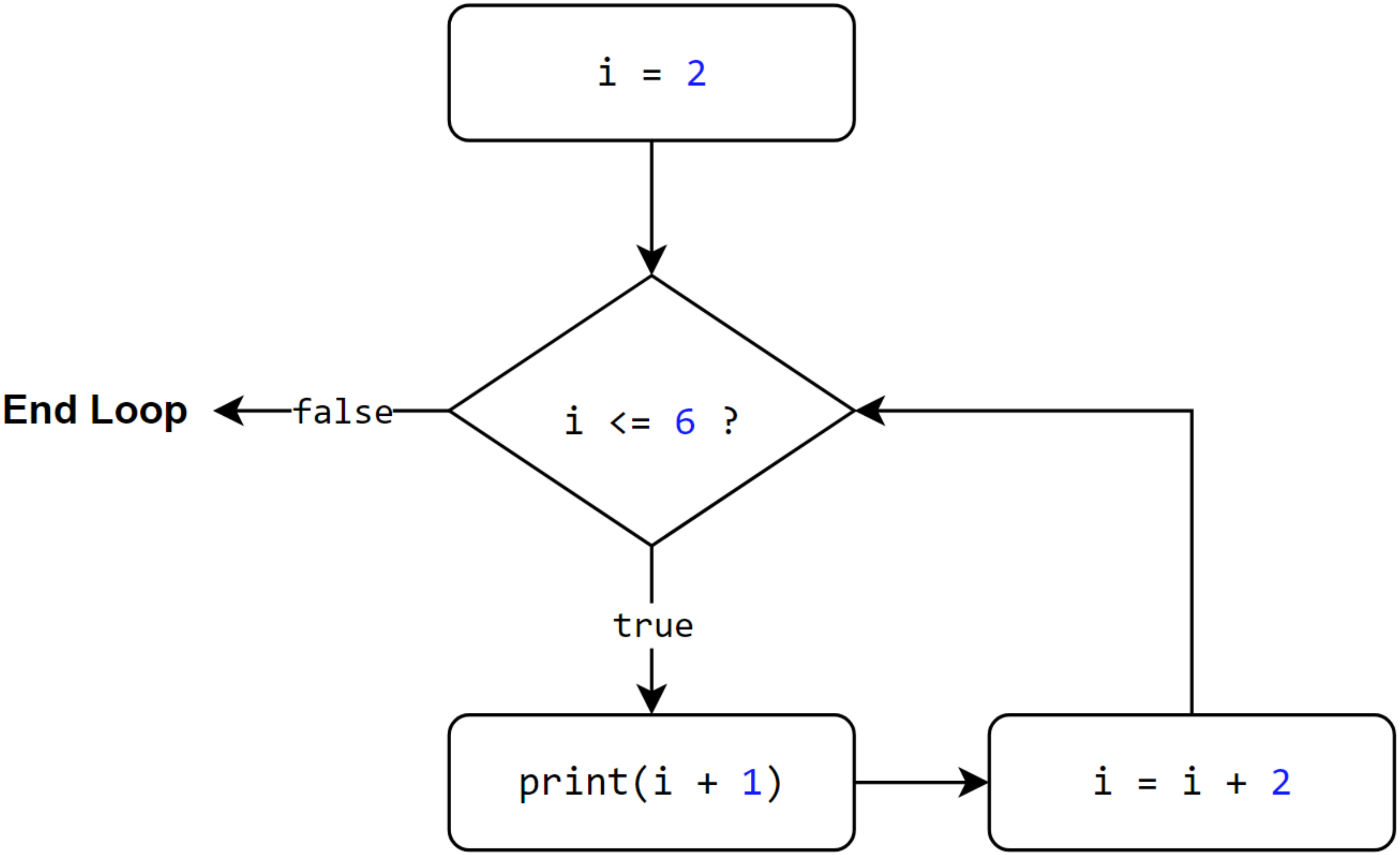
true

Execute loop body

Execute update action

for loop flowchart

```
for (i = 2; i <= 6; i = i + 2) {
    print(i + 1);
}
```



Short-hand operators

Short-hand	Equivalent	Comment
a++	a = a + 1	a++ has the value of a
++a	a = a + 1	++a has the value of (a + 1)
a--	a = a - 1	a-- has the value of a
--a	a = a - 1	--a has the value of (a - 1)
a += 3	a = a + 3	Increments then assigns
a -= 4	a = a - 4	Decrements then assigns
a *= 5	a = a * 5	Multiplies then assigns
a /= 6	a = a / 6	Divides then assigns
a %= 2	a = a % 2	Modulus then assigns

The trace table technique

```
int t;
int x = 3;
for (t = 0; t < 16; t += 3) {
    x *= 3;
}
System.out.println(x);
System.out.println(t);
```

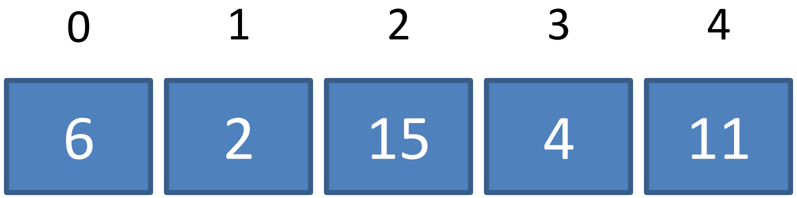
step	t	t < 16	x
1	-	-	-
2	-	-	3
3	0	T	3
4	0	T	9
5	3	T	9
6	3	T	27
7	6	T	27
8	6	T	81
9	9	T	81
10	9	T	243
11	12	T	243
12	…	…	…

The `array` structure

- At times, we have to handle a lot of values and declaring too many variables is not a good option
- So they gave programming languages a tool to group many values into one variable called array
- We can do something like this:

```
int[] a = {6, 2, 15, 4, 11};
System.out.println(a[0] + a[2]); // 6 + 15
```

- We call the values by their **position** in the array
- Position starts from **0**



0	1	2	3	4
6	2	15	4	11

Declare `arrays`

```java
int[] a;
double[] b;
String[] names;
```

- If you declare them like this, they will be **`null`**
- **`null`** is the value of an object which hasn't been initialized yet

```java
String s; // s is null
Scanner sc; // sc is also null
Scanner sc2 = new Scanner(System.in);
// sc2 got initialized and isn't null
sc = new Scanner(System.in);
s = "Hello";
// sc and s are initialized
```

Initialize arrays

```
int[] a = new int[10];
```

- An array of 10 zeros

```
double[] b = new double[5];
```

- A double array of 5 zeros

```
String[] names = new String[3];
```

- An array of 3 null values

→ **Reason:** The default value for int and double is 0 and for String is null

Initialize `arrays` with values

```
int[] a = {2, 4, 6};
```

- An array of 3 numbers

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
```

- An array of 5 real numbers

```
String[] names = {"Ha", "Tu", "Hoa"};
```

- An array of 3 strings

Getting array length

```
int[] a = {2, 4, 6};
System.out.println(a.length); // 3
```

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
System.out.println(b.length); // 5
```

```
String[] names = new String[10];
System.out.println(names.length); // 10
```

→ Array length can be automatically determined (based on initialized values) or specified on declaration

Arrays and the for loop

- Arrays are most useful when combined with the for loop

```
double[] b = {0.2, 0.4, 0.1, -0.13, 0.9};
for (int i = 0; i < b.length; i++) {
   System.out.println("#" + i + ": " + b[i]);
}
```

Result

```
#0: 0.2
#1: 0.4
#2: 0.1
#3: -0.13
#4: 0.9
```

How to stop a `for` loop

- When we search for something with a for loop, we may want to stop looking as soon as it is found.
- E.g. Find one negative number from an array such as: `int[] a = {6,4,-2,6,5,9,15,-6,2};`

```
for (int i = 0; i < a.length; i++) {
   if (a[i] < 0) {
      System.out.println("Found: " + a[i]);
   }
}
```

What is the output of the above piece of code?

How to stop a for loop

```
int[] a = {6, 4, -2, 6, 5, 9, 15, -6, 2};
for (int i = 0; i < a.length; i++) {
   if (a[i] < 0) {
      System.out.println("Found: " + a[i]);
   }
}
```

Output:

```
Found: -2
Found: -6
```

- This piece of code found 2 negative numbers but only one is required.
- After -2 is found at the 3rd iteration, the loop continues to run until it finishes after 9 iterations.
- It should've stopped at the 3rd iteration.

How to stop a for loop

```
int[] a = {6, 4, -2, 6, 5, 9, 15, -6, 2};
for (int i = 0; i < a.length; i++) {
   if (a[i] < 0) {
      System.out.println("Found: " + a[i]);
      break;
   }
}
```

Output:

```
Found: -2
```

- The **break** statement terminates an on-going **for** loop.
- **break** affects the loop which immediately contains it.

How to stop a `for` loop

```java
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (i * j > 30) {
            System.out.println(i + "," + j);
            break; // out of j loop
        }
    }
}
```

Output:

```
4,8
5,7
6,6
7,5
8,4
9,4
```

How to stop a for loop

```java
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 10; j++) {
        if (i * j > 30) {
            System.out.println(i + "," + j);
        }
    }
    if (i == 4) break; // out of i loop
}
```

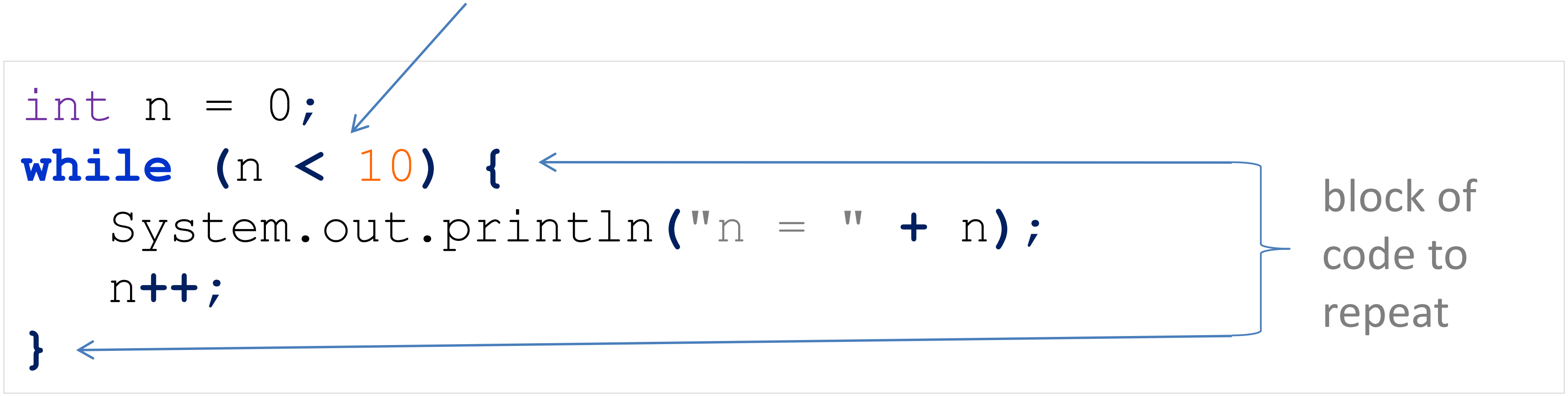
Output:

```
4,8
4,9
```

The while loop

- Repeat a block of code as long as a condition holds true
- The number of iterations is not specific and can be zero

The loop stops when **loop condition** is false



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

block of code to repeat

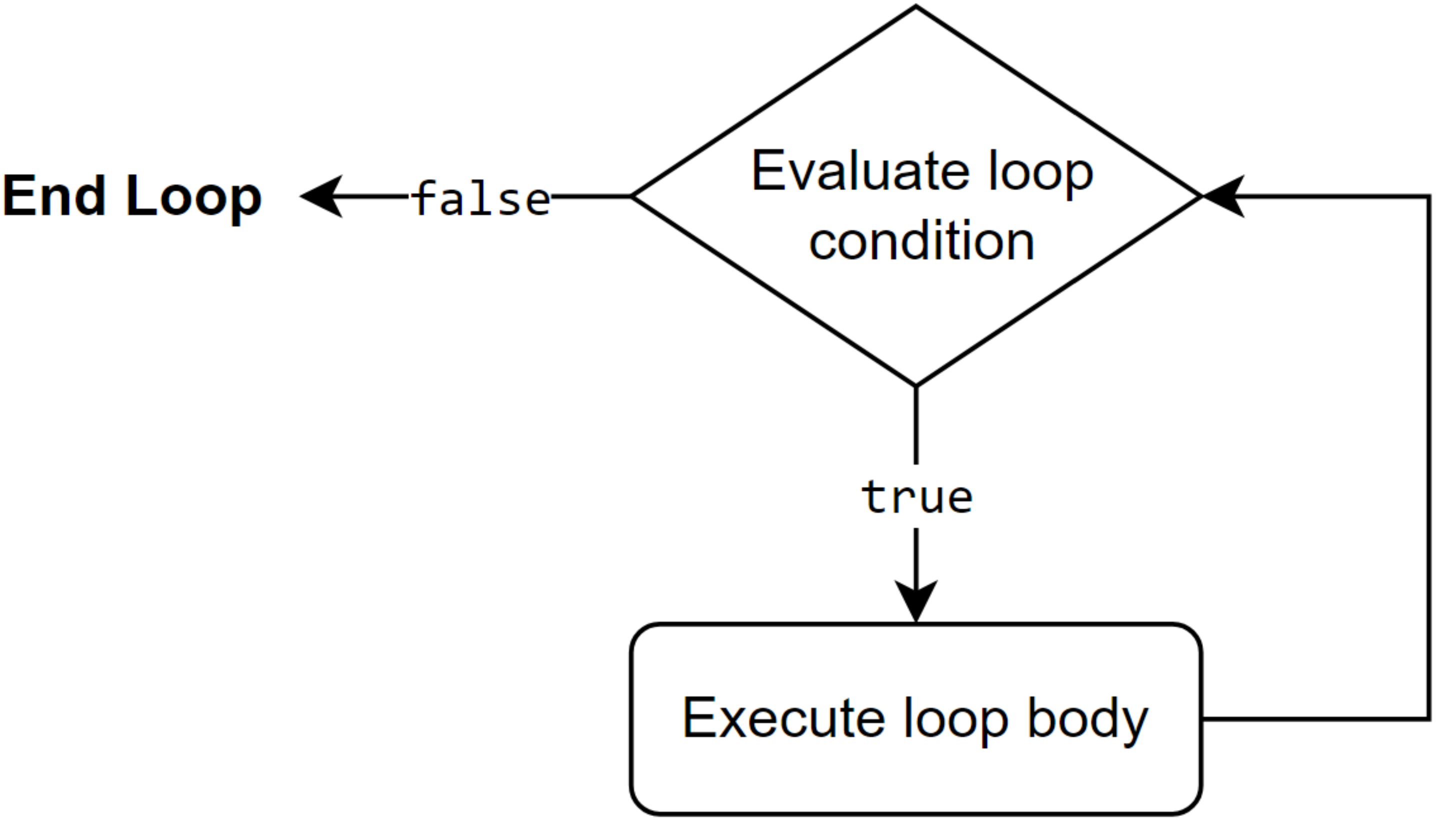
`while` loop explained

```java
int n = 1, e = 0;
while (n < 10) {
    n = n * 2;
    e++;
}
System.out.println("2^" + e + " = " + n);
```

What is the output?

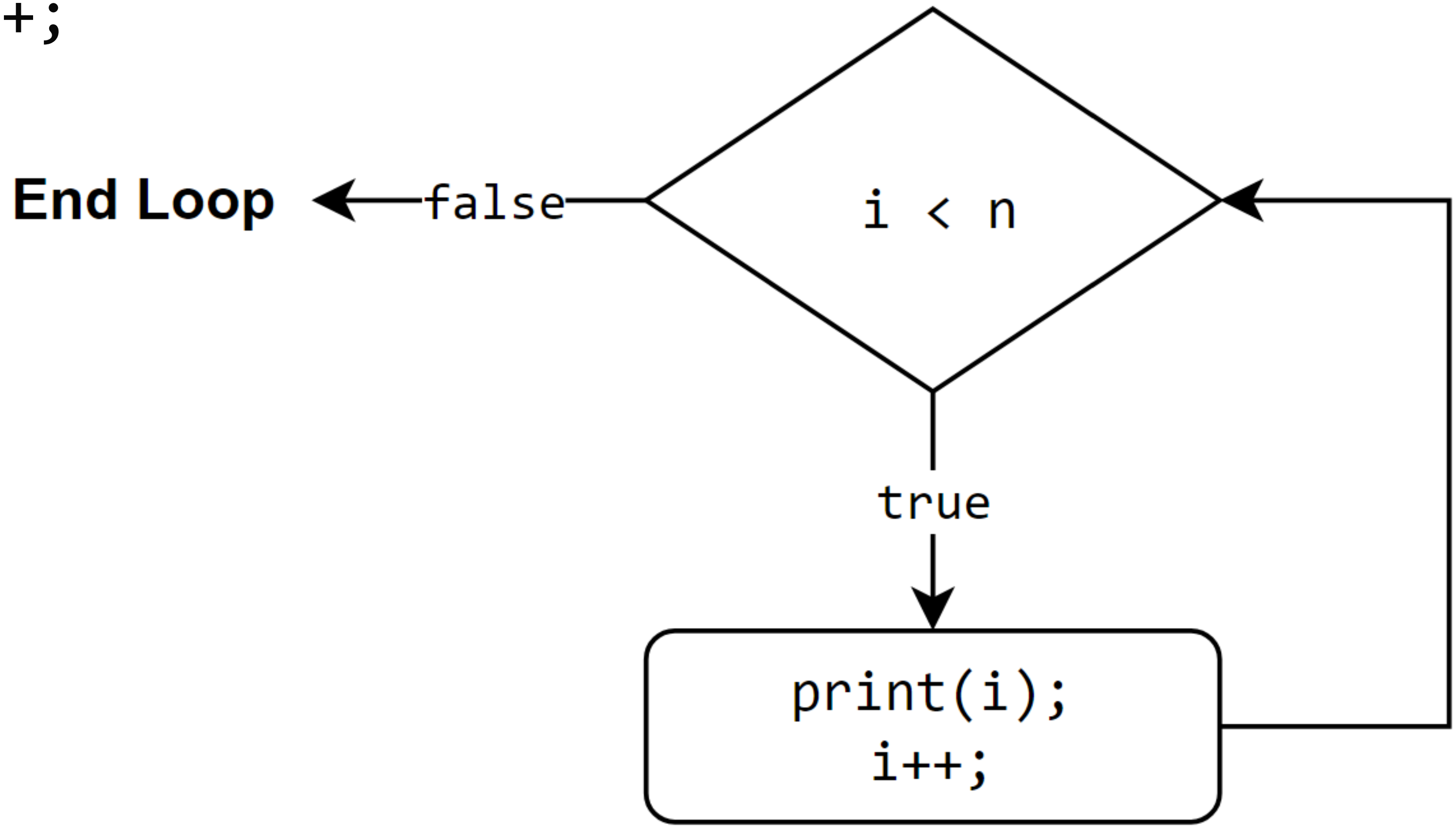
- Let `n = 1, e = 0`
- Now `n < 10` is `true`, let's continue the loop.
- Execute `n = n * 2` and `e++` → `n` becomes `2`, `e` becomes `1`
- The condition `n < 10` is still `true`, let's continue the loop.
- Execute `n = n * 2` and `e++` → `n` becomes `4`, `e` becomes `2`
- The condition `n < 10` is still `true`, let's continue the loop.
- Execute `n = n * 2` and `e++` → `n` becomes `8`, `e` becomes `3`
- The condition `n < 10` is still `true`, let's continue the loop.
- Execute `n = n * 2` and `e++` → `n` becomes `16`, `e` becomes `4`
- Finally `n < 10` is `false`, the loop ends.

while loop flowchart



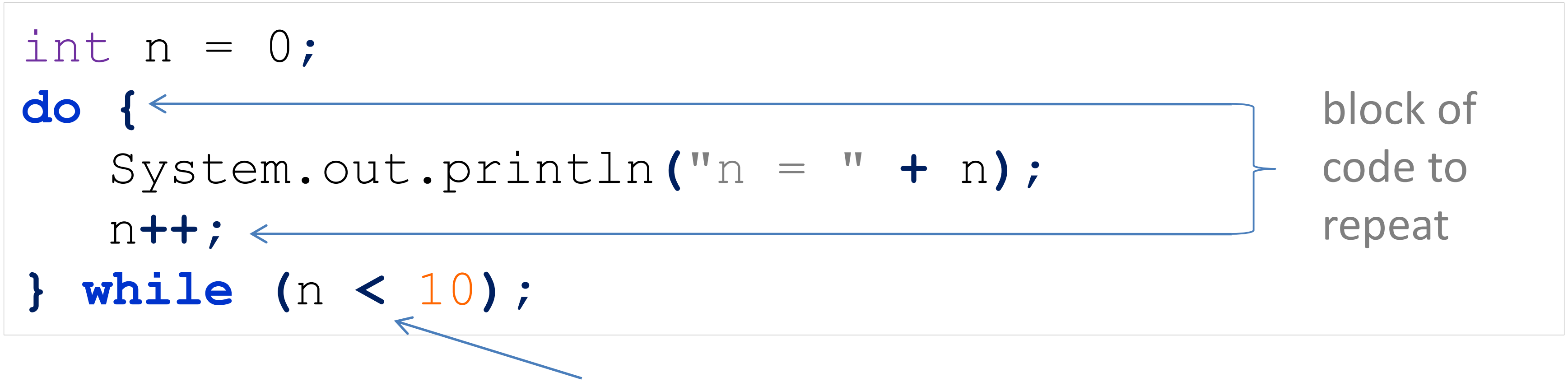
while loop flowchart

```
while (i < n) {
    print(i);
    i++;
}
```



The do…while loop

- Repeat a block of code once, and then continues as long as a condition holds true
- The number of iterations is not specific but always >= 1



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

block of code to repeat

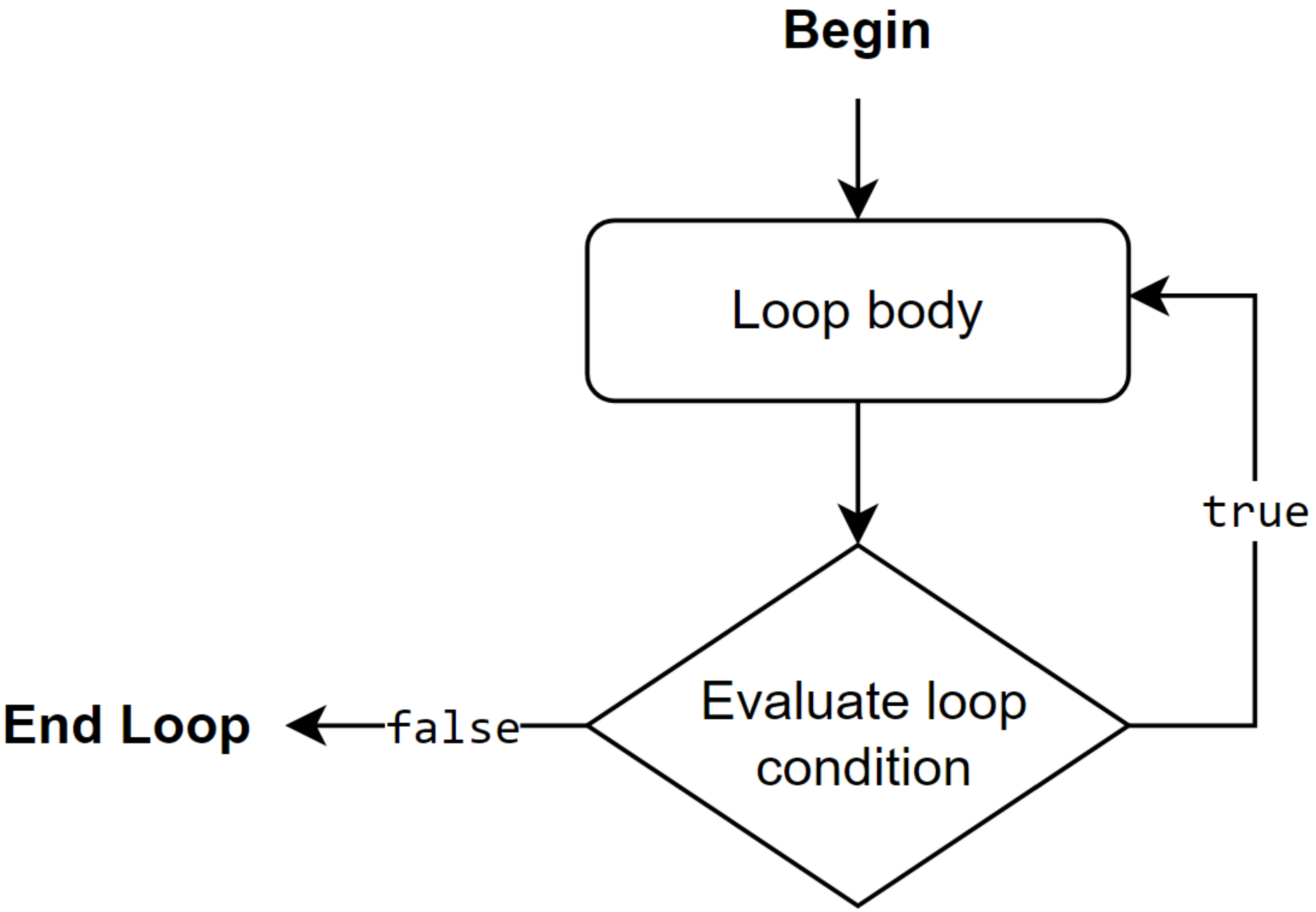
The loop stops when **loop condition** is false

do…while loop explained

```
int n;
do {
   System.out.print("Enter a positive integer: ");
   n = sc.nextInt();
} while (n <= 0);
System.out.println("Thank you!");
```

- Let `n` be uninitialized
- Print a text message to ask user to enter a positive integer.
- Get `n`'s value from the keyboard with `sc.nextInt()` method.
- Repeat if the user does not obey you.

do…while loop flowchart



Begin

Loop body

Evaluate loop condition

true

false

End Loop

Stop a while loop with break

- Similar to the for loop, the while loop can be terminated with the break statement.

```
while (n < 10) {
    if (sc.nextLine().equals("q")) {
        System.out.println("Goodbye!");
        break;
    }
    n++;
}
```

Skip the rest of an iteration with `continue`

- Similar to the `for` loop, an iteration of a `while` loop and do…while loop can be interrupted with `continue`

```
int n = 0;
while (n < 3) {
    n++;
    System.out.println(n);
    if (n == 2) continue;
    System.out.println("...hi");
}
```

Output:

```
1
...hi
2
3
...hi
```

Example

- Replace all spaces in a string with underscores.

Answer 1

```
String s = "To infinity and beyond!";

for (int i = 0; i < s.length(); i++) {
    if (s.charAt(i) == ' ') {
        System.out.print("_");
    } else {
        System.out.print(s.charAt(i));
    }
}

System.out.println(); // add a new line at the end
```

Comment: This solution uses a lot of print statements.

Answer 2

```
String s = "There's a snake in my boot!";
String s2 = "";

for (int i = 0; i < s.length(); i++) {
   if (s.charAt(i) == ' ') {
      s2 = s2 + "_"; // this creates a new String
   } else {
      s2 = s2 + s.charAt(i); // same as above
   }
}

System.out.println(s2);
```

Comment: This solution creates a lot of String objects, which is computationally expensive.

Example

- Calculate the square root of a number without a built-in function (such as `Math.sqrt()`)
- Newton's method of approximation
  - Let the number be N and the desirable square root be S
  - At first, guess that S is 1
  - If S = N / S then S is the square root of N
  - If not, the next guess is the average of S and N / S
  - Continue while the next guess is still different from the previous guess

Answer

```
double n = 50, s = 1, prev_s;
do {
   prev_s = s; // save the previous guess
   System.out.println(prev_s);
   s = (s + n / s) / 2; // update the guess
} while (prev_s != s); // stop if 2 guesses are the same
System.out.println("Result: " + s);
```

Output:

```
1.0
25.5
13.730392156862745
8.685974371897991
7.22119047433115 9
7.072628275743689
7.071067984011346
7.071067811865477
7.0710678118654755
Result: 7.0710678118654755
```