Shortcuts: “for” loop

compact iteration [supp]

Loops revisited

`while` loop is simple and good

only need to consider the condition, when the loop keeps going or stops

other code can contribute to the setup and change in the loop condition

e.g. a counter variable initialised, checked, updated, checked, updated, checked, etc.

```
int counter = 0; // initialise
while ( counter < 10 ) // check condition
{
   counter = counter + 1;  // update something in each iteration
}
```

Most loops follow a general structure, can be shortened as a for loop

`for`

Syntax: `for` (*initialisation* ; *condition* ; *update*) *statement*

"First, do the *initialisation*; then, while the *condition* is true, execute the *statement*. After each execution of the *statement*, execute the *update*."

The four components are called (here, and in the course textbook) *initialisation*, *condition*, *update* and *statement*.

Inside `for`

Here's the order of execution of the `for` loop:

1. *Initialisation* always happens.
2. The *condition* is checked at the beginning of each iteration through the loop: if the condition is false, then the loop isn't executed, and won't be again.
3. The *statement* is executed.
4. The *update* is executed: usually this is incrementing or decrementing an index as you move through an array, but it can be anything you like.

Steps 2-4 are repeated until, when the *condition* is checked at Step 2, it is false.

Initialisation

This is executed *first*, even if the body of the loop is never entered: for instance, given the loop

```
    for (int x = 5; x < 5; x = x + 1) {
        System.out.println("Hah! Made it!");
    }
```

the message is never printed.

Variations on `for`

We can move things around so we can access the index variables outside, like this:

```
int i;
for (i = 0; i < 10; i = i + 1) {
    System.out.println(i);
}
System.out.println(i); // still defined
```

```
int i;
for (i = 10; i >= 0; i = i - 1) {
    System.out.println(i);
}
System.out.println(i); // still defined
```

for example

What will this print?

```java
public class ForExample {
    public static void main(String []  args) {
        int i = 10;
        int j = 0;
        for (j = 1; j < i; j = j + 1) {
            System.out.println("i.j = " + i + "." + j);
            i--;
        }
    }
}
```

for example

What will this print?

```java
public class ForExample {
    public static void main(String [] args) {
        int i = 10;
        int j = 0;
        for (j = 1; j < i; j = j + 1) {
            System.out.println("i.j = " + i + "." + j);
            i--;
        }
    }
}
```

```
~> javac ForExample.java
~> java ForExample
i.j = 10.1
i.j = 9.2
i.j = 8.3
i.j = 7.4
i.j = 6.5
```

More options with for

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

If you miss out the *condition*, it will never be checked.

That means it can never be *false*, so you must have some other way of getting out of the loop or it will go on forever.

Another common(-ish) error with `for`

If you put a semicolon after the parentheses of the `for` loop like this:

```
1       int i = 0;
2       for ( ; i < 10; i = i + 1); {
3           System.out.println("i = " + i);
4       }
```

the code will compile fine.

What will be printed? It's a very subtle error that often occurs and can be hard to spot: the semicolon ';' near the end of line 2 above means *there is no statement to execute in the loop*, so the `System.out.println` command isn't *in* the loop.

The `println` statement will therefore be executed *after* the loop.

What will this loop do?

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

`for` — everything is optional!

You can leave out any or all of the ingredients of the `for` loop:

`while` loop

```
int i = 0;
for ( ; i < 10 ; )
{
    i = i + 1;
}
```

What will it do?