Today:
Java expressions and operators concluded
Java Statements:
  Conditionals: if/then, if/then/else
  Loops: while, for

Next Time: Arrays, methods, program structure, fields vs local variables, public vs private,
the keyword static.

Reading assignments are posted on the web site!

Java Statements

Recall: The core of a Java program is a main method consisting of a sequence of statements followed by semicolons; each statement is executed in sequence, and each has some effect on the state of the computer:

```java
/* File: SampleProgram.java
 * Author: Wayne Snyder (snyder@bu.edu)
 * Date: 2/25/16
 * Purpose: This is a sample problem for lecture 3 in CS 112.
 */

public class SampleProgram {
  public static void main(String[] args) {
    int x; // declare x to be int
    x = 3; // assign value to x
    System.out.println(x);
    double y = 3.4; // combine declaration and assignment
    double z = x + y;
    System.out.println(z);
  }
}
```
Flow of control: conditionals if/then, if/then/else

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:

Simple if/then statement:

Python:
```python
x = 4
print("testing x....")
if( (x % 2) == 0):
    print("x is even")
print("done")
```

Java:
```java
public static void main(String[] args) {
    int x;
    System.out.println("testing x....");
    if( (x % 2) == 0 )
        System.out.println("x is even");
    System.out.println("done");
}
```

Flow of control: conditionals if/then/else

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:

if/then/else statement:

Python:
```python
print("testing x....")
if( (x % 2) == 0):
    print("x is even")
else:
    print("x is odd")
print("done")
```

Java:
```java
public static void main(String[] args) {
    int x = 4;
    System.out.println("testing x....");
    if( (x % 2) == 0 )
        System.out.println("x is even");
    else
        System.out.println("x is odd");
    System.out.println("done");
}
```
Flow of control: conditionals if, if/then, if/then/else

Conditional statements alter the flow of execution by testing some boolean condition and branching one way or the other; you just have to translate what you already know from Python into Java syntax:

**Compound if/then/else statement:**

**Python:**
```python
print("testing x...")
if(x < 0):
    print("x is negative")
elif(x < 10):
    print("x is positive but less than 10")
elif(x < 100):
    print("x is positive but less than 100")
else:
    print("x is greater or equal to 100")
print("done")
```

**Java:**
```java
int x = 4;
System.out.println("testing x...");
if( x < 0 )
    System.out.println("x is negative");
else if( x < 10 )
    System.out.println("x is positive but less than 10");
else if( x < 100 )
    System.out.println("x is positive but less than 100");
else
    System.out.println("x is greater or equal to 100");
System.out.println("done");
```

---

**Flow of control: conditionals if, if/then, if/then/else**

**Compound statements**

Any single statement can be replaced by a compound statement, consisting of a sequence of statements delimited by curly braces, instead of indentation, to indicate that all these statements should be executed in sequence:

**Python:**
```python
x = 4
print("testing x....")
if( ( x % 2 ) == 0):
    print("x is even")
    print("proceeding to divide x by 2");
    x = x / 2;
print("done")
```

**Java:**
```java
int x = 4;
System.out.println("testing x....");
if( x % 2 == 0 )
    System.out.println("x is even");
    System.out.println("proceeding to divide x by 2");
    x = x / 2;
System.out.println("done");
```
Flow of control: compound statements

Compound statements

Any single statement can be replaced by a compound statement, consisting of a sequence of statements delimited by curly braces, instead of indentation, to indicate that all these statements should be executed in sequence:

Python:

```python
print("testing x....")
if (x % 2) == 0:
    print("x is even")
    print("proceeding to divide x by 2")
    x = x / 2
else:
    print("x is odd")
    print("proceeding to add 1 to x")
    x += 1
print("done")
```

Java:

```java
System.out.println("testing x....");
if (x % 2) == 0 {
    System.out.println("x is even");
    System.out.println("proceeding to divide x by 2");
    x = x / 2;
} else {
    System.out.println("x is odd");
    System.out.println("proceeding to add 1 to x");
    x += 1;
}
System.out.println("done");
```

Flow of control: compound statements

**Programming Tip:** I strongly recommend that you ALWAYS use parentheses, even to enclose a single statement:

**Don't:**

```java
if(x < 0)
    System.out.println("x is negative");
else
    System.out.println("x is positive");
```

**Do:**

```java
if(x < 0) {
    System.out.println("x is negative");
}
else {
    System.out.println("x is positive");
}
```
Flow of control: compound statements

**Why?** Because if you decide to add code later, you will avoid a nasty bug (especially if you are used to Python's grouping by indentation):

**What you want:**

```java
if(x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is positive");
    x = x * -1;
}
```

**Not what you want!**

```java
if(x < 0)
    System.out.println("x is negative");
else
    System.out.println("x is positive");
x = x * -1;
```

Will be executed no matter what!

Flow of control: loops **while** and **for**

**while** loop:

Python:

```
x = 6
while(x < 10):
    print(x)
    x += 1
```

Java:

```
x = 6;
while( x < 10 ) {
    System.out.println( x );
    x += 1;
}
```

**for** loop:

Python:

```
for y in range(6,10):
    print(y)
```

Java:

```
for(int y = 6; y < 10; ++y)
    System.out.println(y);
```
Flow of control: loops **while** and **for**

The **for** loop syntax is very different from Python:

- **Initialization**: executed once before loop body
- **Condition**: test at beginning of loop (exactly like **while** loop)
- **Update**: executed once at end of loop body

```java
for(int x = 6; x < 10; ++x) {
    System.out.println(x);
}
```

Flow of control: **break** and **continue**

More complex examples of **for** loops:

- Multiple declarations and multiple updates:

```java
System.out.println("Powers of two:");
for(int x = 0, p = 1; x < 11; ++x, p = p * 2) {
    System.out.println(x + "\t" + p);
}
```

Same as:

```java
int x = 0;
for(int p = 1; x < 11; p = p * 2) {
    System.out.println(x + "\t" + p);
    x ++;
}
```
Flow of control: **break** and **continue**

**break** jumps out of the loop entirely; **continue** jumps to end of loop and continues with next iteration:

Python:
```
x = 0
while(x < 10):
    print(x)
    if(x == 5):
        break
    x += 1
```

Java:
```
x = 0;
while(x < 10 ) {
    System.out.println( x );
    if( x == 5 )
        break;
    x += 1;
}
```

Flow of control: **loops**: break, continue

Note carefully: when executing a **continue** in a **for** loop, the next statement to be executed is the update:

```
for(int x = 6; x < 10; ++x) {
    if(x == 5) {
        x = 7;
        continue;
    }
    System.out.println(x);
} // update ++x executed here
```
Local Variables and Scope

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```java
public static void main(String[] args) {
    System.out.println("Powers of Two");
    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0, x);
        System.out.println(p + " \t " + x);
    }
    int j = 2;
    System.out.println(j);
}
```
Local Variables and Scope

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```java
public static void main(String[] args) {
    System.out.println("Powers of Two");
    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0, x);
        System.out.println(x + "\t" + p)
    }
    int j = 2;
    System.out.println(j);
}
```

Scope of `j`

Local Variables and Scope

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

You may not refer to a variable outside its scope, and you may not re-declare a variable inside its scope.

```java
public static void main(String[] args) {
    System.out.println("Powers of Two");
    int x = 0;
    while(x < 11) {
        int p = Math.pow(2.0, x);
        System.out.println(x + "\t" + p)
    }
    int j = 2;
    System.out.println(j);
}
```

Scope of `p`
Local Variables and Scope

The **scope** of a variable (= where you can refer to it) is from the point of the declaration to the next right curly brace.

Note carefully how this works in a **for** loop – the scope of the initialization variable includes the condition, the update, and the whole loop body:

```java
public static void main(String[] args) {
    System.out.println("Powers of Two");
    for(int i = 0; i < 11; ++i) {
        int p = (int) Math.pow(2.0, i);
        System.out.println(i + "\t" + p);
    }
    System.out.println(i);
}
```

Flow of control: loops: break, continue

This leads us to a useful idiom: when you need to refer to the initialization after the **for** loop, declare it before the loop:

```java
public static void main(String[] args) {
    System.out.println("Powers of Two");
    int i = 0;
    for( ; i < 11; ++i) {
        int p = (int) Math.pow(2.0, i);
        System.out.println(i + "\t" + p);
    }
    System.out.println("Now i has value: " + i);
}
```