Today:
Java Statements:
  conditionals: if and if else
  loops: while and for
Java Data Structures:
  arrays

Next Time: Program structure, methods, and scope of identifiers
Reading assignments are posted on the web site!
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:

Simple if 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");
}
```

Compound if/then 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");
}
```
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 statement:**

```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
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");
```

**Compound statements**

Any branch of a conditional can have multiple statements (called "compound statements" in Java); instead of indentation, we use curly braces to indicate that all these statements should be executed in sequence:

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

Compound statements

Any branch of a conditional can have multiple statements (called “compound statements” in Java); instead of indentation, we use curly braces to indicate that all these statements should be executed in sequence:

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:
```
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: loops: while and for

Loop: while statement

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

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

Loop: for statement

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: break, continue

break and continue work exactly the same as in Python

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

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

```
x = 0
while(x < 10):
    print(x)
    if(x == 5):
        x = 7
        continue
    x += 1
```

Java Data Types: Array

The fundamental data type in Python is a list, which stores a list of values:

Python:
```python
In [1]: A = [2, 3, 4, 6, 7]
In [2]: S = ['hi', 'there', 'folks!']
In [3]: X = [3.14, 3.1415, 3.141592]
In [4]: A[0]
Out[4]: 2
In [5]: S[3]
Traceback (most recent call last):
  File "<ipython-input-5-2cf1e0048e3b>", line 1, in <module>
S[3]
IndexError: list index out of range
```

Java:
```java
In [6]: for i in range(len(X)):
    ...:     print(X[i])
...:
3.14
3.1415
3.141592
```

In [7]:
In Java, the most common way to store a sequence of values is in an array:

Python:
```python
In [1]: A = [2, 3, 4, 6, 7]
In [2]: S = ['hi', 'there', 'folks!']
In [3]: X = [3.14, 3.1415, 3.141592]
In [4]: A[0]
Out[4]: 2
In [5]: S[3]
Traceback (most recent call last):
File "<ipython-input-5-2cfe81dbde3b>", line 1
  S[3]
IndexError: list index out of range
```

Java:
```java
int[] A = {2, 3, 4, 6, 7};
String[] S = {"hi", "there", "folks!"};
double[] X = {3.14, 3.1415, 3.141592};
System.out.println(A[0]);
System.out.println(S[3]);
```

```java
for(int i = 0; i < X.length; ++i)  
  System.out.println(X[i]);
```

In [6]: for i in range(len(X)):
    ...:  print(X[i])
...:
3.14
3.1415
3.141592
```

In [7]:
Java Data Types: Array

In Java, the most common way to store a sequence of values is in an array:

Python:

```python
In [1]: A = [2, 3, 4, 6, 7]
In [2]: S = ['hi', 'there', 'folks!']
In [3]: X = [3.14, 3.1415, 3.141592]
In [4]: A[0]
Out[4]: 2
In [5]: S[3] Traceback (most recent call last):
   File "<ipython-input-5-2cf1e81d4be3>" 5[3]
IndexError: list index out of range
In [6]: for i in range(len(X)): ...
    \ print(X[i])
    ...
3.14
3.1415
3.141592
In [7]:
```

Java:

```java
int[] A = {2, 3, 4, 6, 7};
String[] S = {'hi', 'there', 'folks'};
double[] X = {3.14, 3.1415, 3.141592};
System.out.println(A[0]);
int i = 0; i < X.length; ++i )
    System.out.println(X[i]);
```

As in Python, in Java we can change the value in a particular location in the sequence:

Python:

```python
In [7]: A = [2, 3, 4, 6, 7]
In [8]: A
Out[8]: [2, 3, 4, 6, 7]
In [10]: A
Out[10]: [2, 3, 12, 6, 7]
```

Java:

```java
int[] A = {2, 3, 4, 6, 7};
System.out.println("A["");
for( int i = 0; i < A.length-1; ++i )
    System.out.println( A[i] + "," );
System.out.println( A[A.length - 1] + "]" );
System.out.println("A["");
for( int i = 0; i < A.length-1; ++i )
    System.out.println( A[i] + "," );
System.out.println( A[A.length - 1] + "]" );
```
Java Data Types: Array

But we can NOT change the size of the sequence at run time, as we can in Python:

Python:

```java
In [10]: A
Out[10]: [2, 3, 12, 6, 7]
In [11]: A.append(23)
In [12]: A
Out[12]: [2, 3, 12, 6, 7, 23]
```

Java:

```java
NOT POSSIBLE!
```

The reason has to do with strong typing: in Java, we have to allocate memory for the array when we create it. We can't change the size once this is done. To add to the array, we would have to redo the whole process. Let's look at it from the beginning.....

```java
int[] A = new int[5]; // create a new array of size 5
A[0] = 2; // put values in at run time
A[1] = 3;

System.out.println("\n[ ");
for( int i = 0; i < A.length-1; ++i )
    System.out.print( A[i] + ", " );
System.out.println( A[ A.length - 1 ] + "]" );
```
Warning: You can NOT use an array literal to assign to an array, EXCEPT when you declare it; this is why an array literal is called an “array initializer”:

```java
int[] A = {2, 3, 12, 6, 15}; // OK!
System.out.println("A");
for (int i = 0; i < A.length-1; ++i)
    System.out.print(A[i] + " ");
System.out.println(A[A.length-1] + ");
A = [[2, 3, 12, 6, 15, 23]]; // NOT OK!
```