Announcements & Logistics

- **HW 9** due Mon 12/6 @ 10pm
  - Covers a lot of “advanced” topics from recent lectures
  - Review special methods, iterators, generators, efficiency
  - Good discussion topics for herd meetings
- **Lab 10 Selection Sort in Java** (next Mon/Tue)
  - No pre-lab work or video
  - Hope most of you will start and finish during your lab session
- **Final exam reminder:** Dec 18 @ 9:30 am
- Course evals next Friday 12/10 (bring a laptop to class if possible)

Do You Have Any Questions?
Last Time

- Discussed high level overview of Java vs Python
- Focused on main differences:
  - Java is a **compiled** language: code is compiled before it is executed!
  - Java is **strongly typed**: variables must be declared
- Looked at “Hello World” in Java
  - Broke down all the parts
- Started discussing a simple example which takes input and converts Fahrenheit to Celsius

Java:

```java
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```

```
bash-3.2$ javac Hello.java
bash-3.2$ java Hello
Hello, World!
```
Today

• Break down the simple temperature example further

• Move on to more interesting **data types** in Java
  
  • **Strings**
  
  • **ArrayLists**
  
  • **Arrays**
  
  • **HashMaps**

• Talk about **conditional statements**: very similar to Python!
Recap Simple Example

1. **Consider this Python script:** `temp.py`
2. **What does it do?**
   - Asks user to enter a temperature in Fahrenheit and converts the string input to float
   - Does the computation to convert temperature to Celsius
   - Prints result

```python
1   def main ():
2       fahr = float(input ("Enter the temperature in F: " ))
3       cel = (fahr - 32) * 5.0 / 9.0
4       print ("The temperature in C is: ", cel)
5
6   if __name__ == "__main__":
7       main()
```
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

• Same program in Java: TempConv.java
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

- Same program in Java: `TempConv.java`
Java uses import statements to tell the compiler what classes to use. Java import statements are similar to `from module import xxx` statements in Python.

```
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```
Java is **statically typed**. Thus, all variables must be **declared** with a name and type before they are used. Common convention is to declare variables at the top of our methods/classes.

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

Lines 6-8 are **variable declarations**, which define the name and type of our variables. Once declared, the types cannot be changed.
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: "+ cel);
    }
}
```

- Let’s try to compile: `javac TempConv.java`
The compiler will report several errors (sometimes repeatedly) when we try to compile our program after removing our variable declarations.
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main(String[] args) {
        Double fahr;
        Double cel;
        Scanner in; // On Line 8 we give our Scanner the name in.
        in = new Scanner(System.in); // On Line 10, we initialize our Scanner object with the parameter System.in to read input from the user. Note: Always use new when initializing new objects.
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble();
        System.out.print("The temperature in C is: ");
        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println(" + cel);}
```
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main(String[] args) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner(System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

- `System.out.print` and `System.out.println` are like `print` in Python.
- `in.nextDouble()` automatically reads the user input as a `Double` (like using `input()` in Python and then converting to `float`).
An Aside: Using the Java Scanner

- Since Java is strongly typed, we have to be extra careful when reading input from the user to make sure it is of the expected type.

- The Scanner class provides methods for making sure the next value (like an iterator!) is of the expected type.

- Here are some methods for the Java Scanner class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Computes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextBoolean()</td>
<td>reads and converts next token to a boolean value</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads and converts next token to an integer value</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads and converts next token to a long value</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads and converts next token to a double value</td>
</tr>
<tr>
<td>nextString() or next()</td>
<td>reads next token and returns it as a String</td>
</tr>
<tr>
<td>nextLine()</td>
<td>reads until the next new line and returns a String</td>
</tr>
<tr>
<td>hasNextBoolean()</td>
<td>returns true iff the next token is either “true” or “false”</td>
</tr>
<tr>
<td>hasNextInt()</td>
<td>returns true iff the next token is an integer</td>
</tr>
<tr>
<td>hasNextLong()</td>
<td>returns true iff the next token is a long</td>
</tr>
<tr>
<td>hasNextDouble()</td>
<td>returns true iff the next token is a real number</td>
</tr>
<tr>
<td>hasNextString() or hasNext()</td>
<td>returns true iff there is at least one more token of input</td>
</tr>
<tr>
<td>hasNextLine()</td>
<td>returns true iff there is another line of input</td>
</tr>
</tbody>
</table>
Simple Example

```java
// this is a comment in Java
import java.util.Scanner;

public class TempConv {
    public static void main (String args[]) {
        Double fahr;
        Double cel;
        Scanner in;

        in = new Scanner (System.in);
        System.out.print("Enter the temperature in F: ");
        fahr = in.nextDouble ();

        cel = (fahr - 32) * 5.0 / 9.0;
        System.out.println("The temperature in C is: " + cel);
    }
}
```

- Arithmetic calculations in Java and Python are very similar wrt syntax
- When we print, we use the `+` operator to perform **string concatenation**
Simple Example

bash-3.2$ javac TempConv.java
bash-3.2$ java TempConv
Enter the temperature in F: 98.6
The temperature in C is: 37.0
bash-3.2$ java TempConv
Enter the temperature in F: 32
The temperature in C is: 0.0

• Before running our program, we compile using javac

    javac TempConv.java

• To run, we use java

    java TempConv
Recap: Python vs. Java

Java:

```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

```python
fahr = float(input("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

• Step 1: Prepare to read input from user.
Recap: Python vs. Java

Java:

```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:

```python
fahr = float(input ("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is: ", cel)
```

- Step 2: Prompt user for input.
Step 3: Read user input and convert to float/double (that is, a number with a decimal point).
Recap: Python vs. Java

Java:
```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble () ;
cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:
```python
fahr = float(input("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 4: Perform conversion to Celsius.
Recap: Python vs. Java

Java:
```java
in = new Scanner (System.in);
System.out.print("Enter the temperature in F: ");
fahr = in.nextDouble ();

cel = (fahr - 32) * 5.0 / 9.0;
System.out.println("The temperature in C is: " + cel);
```

Python:
```python
fahr = float(input ("Enter the temperature in F: "))
cel = (fahr - 32) * 5.0 / 9.0
print ("The temperature in C is:", cel)
```

- Step 5: Print result.
An Aside: Java GUIs

- Java has more built-in support for making GUIs and supporting graphical objects
- Here is a graphical version of our program

```java
import javax.swing.*;

public class TempConvGUI {
    public static void main (String args[]) {
        Double fahr, cel;
        String fahrString;

        fahrString = JOptionPane.showInputDialog("Enter the temperature in F: ");
        fahr = Double.valueOf(fahrString);

        cel = (fahr - 32) * 5.0 / 9.0;
        JOptionPane.showMessageDialog(null, "The temperature in C is " + cel);
    }
}
```
Data Type: Strings

• Strings in Java and Python are similar, although the **syntax** is very different.

• Like Python, Strings in Java are also **immutable**.

• Manipulating strings in Java is somewhat less intuitive because Strings **do not support an indexing or slicing operator**.

• You can still index into a Java String by pulling out a **substring**.

• Java uses **method calls** where Python uses operators; Java does not support operator overloading in general.

<table>
<thead>
<tr>
<th>Python</th>
<th>Java</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str[3]</code></td>
<td><code>str.charAt(3)</code></td>
<td>Return character in 3rd position</td>
</tr>
<tr>
<td><code>str[2:5]</code></td>
<td><code>str.substring(2,5)</code></td>
<td>Return substring from 2nd to 4th</td>
</tr>
<tr>
<td><code>len(str)</code></td>
<td><code>str.length()</code></td>
<td>Return the length of the string</td>
</tr>
<tr>
<td><code>str.find('x')</code></td>
<td><code>str.indexOf('x')</code></td>
<td>Find the first occurrence of x</td>
</tr>
<tr>
<td><code>str.split()</code></td>
<td><code>str.split(&quot;&quot;)</code></td>
<td>Split the string on whitespace into a list/array of strings</td>
</tr>
<tr>
<td><code>str.split(',',)</code></td>
<td><code>str.split(',',)</code></td>
<td>Split the string at ',' into a list/array of strings</td>
</tr>
<tr>
<td><code>str + str</code></td>
<td><code>str.concat(str)</code></td>
<td>Concatenate two strings together</td>
</tr>
<tr>
<td><code>str.strip()</code></td>
<td><code>str.trim()</code></td>
<td>Remove any whitespace at the beginning or end</td>
</tr>
<tr>
<td>Java:</td>
<td>Python:</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>String s = &quot;Almost winter break&quot;;</td>
<td>s = &quot;Almost winter break&quot;</td>
<td></td>
</tr>
<tr>
<td>s.substring(0,3) //generate substring</td>
<td>s[::3]</td>
<td></td>
</tr>
<tr>
<td>Alm</td>
<td>'Alm'</td>
<td></td>
</tr>
<tr>
<td>s.substring(4,7) //generate substring</td>
<td>s[4:7]</td>
<td></td>
</tr>
<tr>
<td>st</td>
<td>'st'</td>
<td></td>
</tr>
</tbody>
</table>
Strings

Java:

String s = "Almost winter break";

s.substring(0,3) //generate substring
Alm

s.substring(4,7) //generate substring
st

s.toUpperCase() //convert s to upper case
ALMOST WINTER BREAK

s.toLowerCase() //convert s to lower case
almost winter break

Syntax to print an array
String [] array = s.split(" ") //split s
System.out.println(Arrays.toString(array))
[Almost, winter, break]

Python:

s = "Almost winter break"

s[:3]
'Alm'

s[4:7]
'st'

s.upper()
'ALMOST WINTER BREAK'

s.lower()
'almost winter break'

print(array)
['Almost', 'winter', 'break']
Data Type: ArrayLists

• **ArrayLists** in Java are roughly equivalent to **Lists** in Python

• Both are basically **dynamic arrays** (that grow and shrink in size automatically)

• However, unlike Python where lists can contain anything, in Java we declare what **type** of objects our **ArrayList** is going to contain

• We **cannot use** [*] operator **“list notation”** in **ArrayLists**

• Instead rely on **methods** (like `get()`, `set()`, `add()`) to manipulate the list instead

• We will first compare **ArrayLists** to **lists** in Python

• Next, we will discuss a primitive data structure **Arrays** which are also analogous to Python **lists** but are statically-sized, more commonly used, and do support [*] operator **“list notation”**
ArrayLists vs. Lists

Java:

```java
ArrayList<String> alist = new ArrayList<String>();
alist.add("Jeannie");
alist.add("Shikha");
alist.add("Lida");
alist.add("Kelly");
alist.add("Duane");
alist.add("Aaron");
true
System.out.println(alist)  // print the list

[Jeannie, Shikha, Lida, Kelly, Duane, Aaron]
```

Python:

```python
alist = []
alist.append("Jeannie")
alist.append("Shikha")
alist.append("Lida")
alist.append("Kelly")
alist.append("Duane")
alist.append("Aaron")
print(alist)

['Jeannie', 'Shikha', 'Lida', 'Kelly', 'Duane', 'Aaron']
```
ArrayLists vs. Lists

Java:

```java
ArrayList<String> alist = new ArrayList<String>();
alist.add("Jeannie");
alist.add("Shikha");
alist.add("Lida");
alist.add("Kelly");
alist.add("Duane");
alist.add("Aaron");

true

System.out.println(alist) // print the list
[Jeannie, Shikha, Lida, Kelly, Duane, Aaron]

alist.add(3, "Rohit") // add Rohit to index 3

System.out.println(alist)
[Jeannie, Shikha, Lida, Rohit, Kelly, Duane, Aaron]

alist.get(2) // get the element at index 2
Lida

alist.set(5, "Steve") // set index 5 to Steve
Duane

System.out.println(alist)
[Jeannie, Shikha, Lida, Rohit, Kelly, Steve, Aaron]
```

Python:

```python
alist = []
alist.append("Jeannie")
alist.append("Shikha")
alist.append("Lida")
alist.append("Kelly")
alist.append("Duane")
alist.append("Aaron")

print(alist)
['Jeannie', 'Shikha', 'Lida', 'Kelly', 'Duane', 'Aaron']

alist.insert(3, "Rohit")

print(alist)
['Jeannie', 'Shikha', 'Lida', 'Rohit', 'Kelly', 'Duane', 'Aaron']

alist[2]
'Lida'

alist[5] = "Steve"

print(alist)
['Jeannie', 'Shikha', 'Lida', 'Rohit', 'Kelly', 'Steve', 'Aaron']
```
Data Type: Arrays

- An **array** is a primitive data structure in Java (with corresponding **Arrays** objectified class), and are also similar to **Lists**
- They do **support list notation**
- They **cannot dynamically shrink and grow**
- To declare a new array object in Java, we need to specify the **type** of its values and the **size** it will have
  - Size must be **specified directly**, or
  - Can just **assign values** at declaration
- Unlike lists in Python we cannot store heterogeneous types in an array!
Data Type: Arrays

- An **array** is a primitive data structure in Java.
- Can use list notation and assign values directly (but cannot grow or shrink).

```java
import java.util.Arrays;

public class Test {

    public static void main(String args[]) {
        int size = 10;

        // option 1: create an array directly
        int[] array1 = new int[] {2, 3, 5};

        // option 2: declare an with size then assign values
        int[] array2 = new int[3];
        array2[0] = 2;
        array2[1] = 3;
        array2[2] = 5;

        System.out.println(Arrays.toString(array1));
        System.out.println(Arrays.toString(array2));
    }
}
```

Declaring a statically-sized array by initializing it with values.

 Declare empty array with size and then assign values later.
Java Arrays: More Examples

When declaring, either define size or give specific values. (Not necessary to do both!)

<table>
<thead>
<tr>
<th>In [1]:</th>
<th>import java.util.Arrays;</th>
</tr>
</thead>
<tbody>
<tr>
<td>In [3]:</td>
<td>String [] myList = new String[6];</td>
</tr>
<tr>
<td>In [4]:</td>
<td>String [] myList = {&quot;Jeannie&quot;, &quot;Shikha&quot;, &quot;Lida&quot;, &quot;Kelly&quot;, &quot;Duane&quot;, &quot;Aaron&quot;};</td>
</tr>
<tr>
<td>In [5]:</td>
<td>System.out.println(Arrays.toString(myList));</td>
</tr>
<tr>
<td></td>
<td>[Jeannie, Shikha, Lida, Kelly, Duane, Aaron]</td>
</tr>
<tr>
<td>In [6]:</td>
<td>System.out.println(myList[2]);</td>
</tr>
<tr>
<td></td>
<td>Lida</td>
</tr>
<tr>
<td>In [7]:</td>
<td>myList[5] = &quot;Steve&quot;</td>
</tr>
<tr>
<td>Out[7]:</td>
<td>Steve</td>
</tr>
<tr>
<td>In [8]:</td>
<td>System.out.println(Arrays.toString(myList));</td>
</tr>
<tr>
<td></td>
<td>[Jeannie, Shikha, Lida, Kelly, Duane, Steve]</td>
</tr>
</tbody>
</table>

Java provides an array wrapper class called `Arrays` that provides convenient static methods for working with primitive arrays.

Can use list notation

Can replace values, but can't easily insert
Other Data Types: Dictionaries

- **HashMaps** in Java are roughly equivalent to **Dictionaries** in Python.
- Both provide easy (O(1)) access to key, value pairs.
- Both provide convenient methods for interacting with/manipulating the keys, values efficiently.
- Both require keys to be unique.
- Java **HashMaps** do not support `[ ]` operator.
  - Must use methods (like `put()`, `get()`, `containsKey()`) to manipulate **HashMap**.
- Python **Dictionaries** support `[ ]` operator and methods.
### HashMaps vs. Dictionaries

**Java:**

```java
HashMap<Integer, String> csCourses;
csCourses = new HashMap<Integer, String>();
csCourses.put(237, "Computer Organization");
csCourses.put(134, "Intro to Computer Science");
csCourses.put(136, "Data Structures");
csCourses.put(256, "Algorithms");
```

- `csCourses.get(237)`
  - `Computer Organization`

- `csCourses.get(134)`
  - `Intro to Computer Science`

- `csCourses.containsKey(134)`
  - true

- `csCourses.containsKey(361)`
  - false

- `csCourses.containsValue("Data Structures")`
  - true

**Python:**

```python
csCourses = dict()
csCourses[237] = "Computer Organization"
csCourses[134] = "Intro to Computer Science"
csCourses[136] = "Data Structures"
csCourses[256] = "Algorithms"
```

- `csCourses[237]`
  - 'Computer Organization'

- `csCourses.get(134)`
  - 'Intro to Computer Science'

- `134 in csCourses`
  - True

- `361 in csCourses.keys()`
  - False

- `"Data Structures" in csCourses.values()`
  - True
Programming Language Features

- **Basic features:**
  - Data Types
  - Reading user input
  - Loops
  - **Conditionals**

- **Advanced topics:**
  - Classes
  - Interfaces
  - Collections
  - Graphical User Interface Programming
Booleans

- **Boolean** (or **boolean**) values in Java:
  - **true** and **false** (no capitalization)
  - Example: `Boolean b = true`

- **Boolean** operators in Java:
  - `&&` - and
  - `||` - or
  - `!` - not
  - Most other operators (`<`, `>`, `==`, etc) are the same as Python
Conditional Statements

- **Conditional** (if-else) statements in Python and Java are very similar.
- Let’s consider three basic patterns

1. Simple if in Python:

```python
if condition:
    statement1
    statement2
    ...
    ...
```

2. Simple if in Java:

```java
if (condition) {
    statement1;
    statement2;
    ...
    ...
}
```
Conditional Statements

- **Conditional** (if-else) statements in Python and Java are very similar.
- Let's consider three basic patterns.

2. if else in Python:

   ```python
   if condition:
       statement1
       statement2
       ...
   else:
       statement1
       statement2
       ...
   ```

   if else in Java:

   ```java
   if (condition) {
       statement1;
       statement2;
       ...
   } else {
       statement1;
       statement2;
       ...
   }
   ```

Note the use of () and {}
Conditional Statements

- **Conditional** (if-else) statements in Python and Java are very similar.
- Let's consider three basic patterns.

3. if elif else in Python:

```python
if condition:
    statement1
    statement2
    ...
elif condition:
    statement1
    statement2
    ...
else:
    statement1
    statement2
    ...
```

Nested if else if in Java:

```java
if (condition) {
    statement1;
    statement2;
    ...
} else if (condition) {
    statement1;
    statement2;
    ...
} else {
    statement1;
    statement2;
    ...
}
```

Java does not have an elif equivalent.
Conditional Statements

Java:

```java
int a = 1;
int b = 2;
if (a < b) {
    System.out.println("a < b");
}

a < b

if (a > b) {
    System.out.println("a > b");
} else {
    System.out.println("a < b");
}

a < b

int c = 3;
if (a > b && a > c) {
    System.out.println("a is largest");
} else if (b > a && b > c) {
    System.out.println("b is largest");
} else {
    System.out.println("c is largest");
}

c is largest
```

Python:

```python
a = 1
b = 2
if a < b:
    print("a < b")

a < b

if a > b:
    print("a > b")
else:
    print("a < b")
a < b

c = 3
if a > b and a > c:
    print("a is largest")
elif b > a and b > c:
    print("b is largest")
else:
    print("c is largest")
c is largest
```
Lecture 5 Revisited

- Recall one of the first examples we looked at involving conditionals in Python (slightly modified to accept user input)

```python
1 def weather(temp):
2     if temp > 80:
3         print("It is a hot one out there.")
4     elif temp >= 60:
5         print("Nice day out, enjoy!")
6     elif temp >= 40:
7         print("Chilly day, wear a sweater.")
8     else:
9         print("Its freezing out, bring a winter jacket!")
10
11 if __name__ == "__main__":
12     temp = int(input("Enter temp: "))
13     weather(temp)
```
Lecture 5 Revisited

- Let’s write it in Java!
Lecture 5 Revisited

```java
import java.util.Scanner;

public class WeatherFinal {
    public static void main (String args[]) {
        int temp;
        Scanner in;

        in = new Scanner(System.in);
        System.out.print("Enter temp: ");
        temp = in.nextInt();

        if (temp > 80) {
            System.out.println("It is a hot one out there.");
        } else if (temp >= 60) {
            System.out.println("Nice day out, enjoy!");
        } else if (temp >= 40) {
            System.out.println("Chilly day, wear a sweater.");
        } else {
            System.out.println("Its freezing out, bring a winter jacket!");
        }
    }
}
```