Announcements

- Lab 1 is tomorrow! Don't forget to do your pre-lab...

- HW1 posted, due next Wednesday
C++ versus Python

High level versus low level
Interpreter versus compiler
Dynamic versus static typing

In Python:
\[
\begin{align*}
a &= 10 \\
a &= "hello"
\end{align*}
\]
Gives an error in C++

\[
\begin{align*}
\text{int} &\quad a, j \\
a &= 10 \\
a &= "hello" \quad \text{syntax error} \\
a &= 'a'j \quad \text{(no error)} \quad \text{give integer ASCII+}
\end{align*}
\]
Why learn C++?

- Efficiency
- Ubiquitous
- Low level
- Complex
- Useful
Variables

Numerical: short, int, long, float, double

bool - true, false

char 'a' (ASCII value)

string "word"
Mutable versus immutable

**Dfn:** mutable - allowed to change

Ex from python: list, dictionary

**Dfn:** immutable - can’t change

Ex: tuples, strings

```
word [1] = 'a'
ERROR (in python)
```
C++ - Maximum Flexibility

In C++, everything is mutable!

```cpp
string word;
word = "Hello";
word[0] = "J";
```

(in C++, no error)

So be careful!
Arrays

Python has lists, tuples, etc.

C++ only has arrays.
- size is fixed
- type is fixed (1 homogenous)

Ex: `int numbers [10];`
numbers [0] = 56;
numbers [9] = 11;

Numbers [10] = 5; Error!
Creating variables (cont.)

Allowed:
\[
\text{int daysInMonth[]} = \{31, 28, 31, 30, 31, 31, 30, 31, 30, 31\}.
\]

Error:
\[
\text{int daysInMonth[]} ;
\]

I need a size

Allowed:
\[
\text{char greeting[]} = \text{"Hello"} ;
\]

\[
\text{int size = 12} ;
\]

\[
\text{int daysInMonth [size]} ;
\]
Creating variables - a few examples

```java
int number;
int a, b; // creates 2 integers.
int age(40);
int age (curYear - birthYear);
int age(40), zipcode(63116);
String greeting("Hello");
```
Forcing things to be immutable:

In some situations, there will be data that we want to be fixed.

To do this, use const:

```javascript
const float gravity(9.8);
```

forces value to be same

```javascript
gravity = 10.0; // Error
```
Operators

Basic numeric operators differ slightly:

<table>
<thead>
<tr>
<th>Arithmetic Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Python</strong></td>
</tr>
<tr>
<td>$-a$</td>
</tr>
<tr>
<td>$a + b$</td>
</tr>
<tr>
<td>$a - b$</td>
</tr>
<tr>
<td>$a * b$</td>
</tr>
<tr>
<td>$a % b$</td>
</tr>
<tr>
<td>$a + b$</td>
</tr>
<tr>
<td>$a / b$</td>
</tr>
<tr>
<td>$a // b$</td>
</tr>
<tr>
<td>$++a$</td>
</tr>
<tr>
<td>$a++$</td>
</tr>
<tr>
<td>$--a$</td>
</tr>
<tr>
<td>$a--$</td>
</tr>
</tbody>
</table>

```c
int a, b;
float c;
c = float(a) / float(b);
OK
```
### Boolean operators & comparators - very different

<table>
<thead>
<tr>
<th>Python</th>
<th>C++</th>
</tr>
</thead>
</table>

### Boolean Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>C++</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>and</code></td>
<td><code>&amp;&amp;</code></td>
<td>logical and</td>
</tr>
<tr>
<td><code>or</code></td>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>not</code></td>
<td><code>!</code></td>
<td>logical negation</td>
</tr>
<tr>
<td><code>a if b else c</code></td>
<td><code>b ? a : c</code></td>
<td>conditional expression</td>
</tr>
</tbody>
</table>

### Comparison Operators

<table>
<thead>
<tr>
<th>Comparison</th>
<th>C++</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a &lt; b</code></td>
<td><code>a &lt; b</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>a &lt;= b</code></td>
<td><code>a &lt;= b</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>a &gt; b</code></td>
<td><code>a &gt; b</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>a &gt;= b</code></td>
<td><code>a &gt;= b</code></td>
<td>greater than or equal to</td>
</tr>
<tr>
<td><code>a == b</code></td>
<td><code>a == b</code></td>
<td>equal</td>
</tr>
<tr>
<td><code>a &lt; b &lt; c</code></td>
<td><code>a &lt; b &amp;&amp; b &lt; c</code></td>
<td>chained comparison</td>
</tr>
</tbody>
</table>
Converting between types:
Be careful! C++ cares about type

```cpp
int a(5);
double b = a;  \leftarrow 5.0
```

```cpp
int a;
double b(2.67);
a = b; \leftarrow 2.67
```

(can’t go between strings & #s at all although chars are given their ASCII value)
Control Structures

C++ has loops, conditionals, functions, and objects.

Syntax is similar — but usually just different enough to get you into trouble, also...
While loops

while (bool)

3

body;

Note:
- `bool` is any boolean exp: `a < b`
- don't need `3` if only one command in body:

\[
\text{while} (a < b) \\
\text{a++}
\]
Also have do-while:

```cpp
int number;
do {
    cout << "Enter a number from 1 to 10: ";
    cin >> number;
} while (number < 1 || number > 10);
```

This is a bit different:

- body of loop is executed once, before repeated condition is checked.
Conditionals

```
if (bool)
{
    body 1;
}
else
{
    body 2;
}
else if (3)
```

Ex: if (x < 0) x = -x;

Note: - don't need brackets if only one line in body
- don't need else
- no elseif in C++ - write out else if
Boolean conditionals in if & while statements

If statements can also be written with numeric conditions instead of booleans:

Ex: if (mistakeCount) cout << "There were " << mistakeCount << " problems" << endl;

if not = 0, true
0 always false
Common mistake - what is wrong?

```cpp
double gpa;
cout << "Enter your gpa: ";
cin >> gpa;
if (gpa == 4.0)
    cout << "Wow!" << endl;
```

In Python, get an error
In C++, sets gpa to 4.0
For loops

Example:

for (int count = 10; count > 0; count --) {
    cout << count << endl;
    cout << "Blastoff!" << endl;
}

Note: int declaration isn't required.

Alternate:

    int count;
    for (count = 10; count > 0; count --) {
        cout << count << endl;
    }
Defining a function: example

```cpp
void countdown() {
    for (int count = 10; count > 0; count--)
        cout << count << endl;
}
```

Remember our countdown function from 150?

Or with optional parameters:

```cpp
void countdown(int start=10, int end=1) {
    for (int count = start; count >= end; count--)
        cout << count << endl;
}
```

More on functions in lab tomorrow...