Announcements

- Intro to Linux lab today at 4pm
- Lab 1 is posted (actually, all labs are posted)
- HW1 is posted, due 1 week from Sat.
- Transition guide is posted

door code: 80386
Comparison

Python

```python
def gcd(u, v):
    # we will use Euclid's algorithm
    # for computing the GCD
    while v != 0:
        r = u % v  # compute remainder
        u = v
        v = r
    return u

if __name__ == '__main__':
    a = int(input('First value: '))
    b = int(input('Second value: '))
    print('gcd:', gcd(a, b))
```

C++

```cpp
#include <iostream>
using namespace std;

int gcd(int u, int v) {
    /* We will use Euclid's algorithm
       for computing the GCD */
    int r;
    while (v != 0) {
        r = u % v;  // compute remainder
        u = v;
        v = r;
    }
    return u;
}

int main() {
    int a, b;
    cout << "First value: ";
    cin >> a;
    cout << "Second value: ";
    cin >> b;
    cout << "gcd: " << gcd(a, b) << endl;
    return 0;
}
White space - returns, tabs, etc. are ignored in C++.

```c
int gcd(int u, int v) { int r; while (v != 0) { r = u % v; u = v; v = r; } return u; }
```

This is not acceptable to submit.

(Recall that these were very important in Python)

Here, we use () and {} to mark loops, booleans, etc.
Compiling

- In Python, you save code as `gcd.py` and then type `"python gcd.py"` to run it.

  Later: `makefile`

- In C++:
  * Save as `gcd.cpp`
  * Type `g++ -o gcd gcd.cpp` (C++ compiler output file)
  * Type `./gcd` run in this directory

  Optional: without it, executable is called `a.out`
## Data Types

<table>
<thead>
<tr>
<th>C++ Type</th>
<th>Description</th>
<th>Literals</th>
<th>Python analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>logical value</td>
<td>true, false</td>
<td>bool</td>
</tr>
<tr>
<td>short</td>
<td>integer (often 16 bits)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int</td>
<td>integer (often 32 bits)</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>long</td>
<td>integer (often 32 or 64 bits)</td>
<td>39L</td>
<td>int</td>
</tr>
<tr>
<td>—</td>
<td>integer (arbitrary-precision)</td>
<td></td>
<td>long</td>
</tr>
<tr>
<td>float</td>
<td>floating-point (often 32 bits)</td>
<td>3.14f</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>floating-point (often 64 bits)</td>
<td>3.14</td>
<td>float</td>
</tr>
<tr>
<td>char</td>
<td>single character</td>
<td>'a'</td>
<td></td>
</tr>
<tr>
<td>string$^a$</td>
<td>character sequence</td>
<td>&quot;Hello&quot;</td>
<td>str</td>
</tr>
</tbody>
</table>

$^a$ import string
Data Types (cont)

- Ints can also be unsigned:
  instead of ranging from $-2^{b-1}$ to $2^{b-1} - 1$
  go from 0 to $2^{b-1}$.

- Strings and chars are very different.
Char versus string.

```cpp
char a;
a = 'a';
a = 'h';

string word;
word = "CS 180";
```

Strings are not automatically included. Standard in most libraries, but need to import.
<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.size()</td>
<td>Either form returns the number of characters in string s.</td>
</tr>
<tr>
<td>s.length()</td>
<td></td>
</tr>
<tr>
<td>s.empty()</td>
<td>Returns true if s is an empty string, false otherwise.</td>
</tr>
<tr>
<td>s[index]</td>
<td>Returns the character of string s at the given index (unpredictable when index is out of range).</td>
</tr>
<tr>
<td>s.at(index)</td>
<td>Returns the character of string s at the given index (throws exception when index is out of range).</td>
</tr>
<tr>
<td>s == t</td>
<td>Returns true if strings s and t have same contents, false otherwise.</td>
</tr>
<tr>
<td>s &lt; t</td>
<td>Returns true if s is lexicographical less than t, false otherwise.</td>
</tr>
<tr>
<td>s.compare(t)</td>
<td>Returns a negative value if string s is lexicographical less than string t, zero if equal, and a positive value if s is greater than t.</td>
</tr>
<tr>
<td>s.find(pattern)</td>
<td>Returns the least index (greater than or equal to index pos, if given), at which pattern begins; returns stringnpos if not found.</td>
</tr>
<tr>
<td>s.find(pattern, pos)</td>
<td></td>
</tr>
<tr>
<td>s.rfind(pattern)</td>
<td>Returns the greatest index (less than or equal to index pos, if given), at which pattern begins; returns stringnpos if not found.</td>
</tr>
<tr>
<td>s.rfind(pattern, pos)</td>
<td></td>
</tr>
<tr>
<td>s.find_first_of(charset)</td>
<td>Returns the least index (greater than or equal to index pos, if given), at which a character of the indicated string charset is found; returns stringnpos if not found.</td>
</tr>
<tr>
<td>s.find_first_of(charset, pos)</td>
<td></td>
</tr>
<tr>
<td>s.find_last_of(charset)</td>
<td>Returns the greatest index (less than or equal to index pos, if given), at which a character of the indicated string charset is found; returns stringnpos if not found.</td>
</tr>
<tr>
<td>s.find_last_of(charset, pos)</td>
<td></td>
</tr>
<tr>
<td>s + t</td>
<td>Returns a concatenation of strings s and t.</td>
</tr>
<tr>
<td>s.substr(start)</td>
<td>Returns the substring from index start through the end.</td>
</tr>
<tr>
<td>s.substr(start, num)</td>
<td>Returns the substring from index start, continuing num characters.</td>
</tr>
<tr>
<td>s.cstr( )</td>
<td>Returns a C-style character array representing the same sequence of characters as s.</td>
</tr>
</tbody>
</table>
Mutable versus immutable

**Def:** mutable : you can change it
- list

**Def:** immutable : you can't change it
- strings in Python
- tuples

In C++, everything is mutable.
C++: Maximum flexibility

Everything is mutable by default!

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

> word = "Jello"
Creating variables

All variables must be explicitly created and given a type.

```java
int number;
int a, b;  // not char a, int b;
int age (35);
int age2 (currYear - birthYear);
int age3 (21), zipcode (63116);
String greeting ("Hello");
```
Immutable variables

We can force some variables to be immutable—use const:

```cpp
const float gravity (-9.8);
```

Why? - don't allow changes
Converting between types

Be careful!

```cpp
int a(5);
double b;
b = a;               // b is 5.0
```

```cpp
int a;
double b (2.67);
a = b;           // a = 2
```
Converting with strings

- Can't go between strings & numeric types at all.
  \[ \text{int } x = "37". \]

- But chars will convert to numbers.
  \[ \text{ASCII} \]
Control Structures

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

Syntax is similar, but just different enough to get into trouble.

(remember to use your book's index in a pinch!)
While loops

while (bool) & body:

Notes:
- bool is any boolean expression
- don't need & if only 1 command in the loop: while (a < b) a++;

→ while (bool) & body:?
**Booleans**

Python

C++

```c
while (! (a == b))
```

<table>
<thead>
<tr>
<th>Boolean Operators</th>
<th>Logical Operators</th>
<th>Meaning</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 cond else b</code></td>
<td><code>cond ? a : b</code></td>
<td>conditional expression</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comparison Operators</th>
<th>Logical Operators</th>
<th>Meaning</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>
Defining a function: example

Remember countdown function from 150?

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

- Functions go at top, before main.
Optional arguments

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

void myfun(int a, int b) {
    cout << a;
}
```
If statements

if (bool) 
    body 1;
else if
    body 2;

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

if (groceries.length() > 15)
    cout << "Go to the grocery store" << endl;
else if (groceries.contains("milk")
    cout << "Go to the convenience store" << endl;

Note:
- Don't need brackets if 1 line
- Don't need else
- No else if
Booleans & if/whiles

If & while statements can be written with numeric conditions (which are really booleans).

Ex: if (mistakeCount)
    cout << "Error!" << endl;

0 => false

anything else => true
Try accounts
+ change passwords!

get console
type passwd