Class Objectives.

• What are we going to do in this class?

  Compare and contrast different programming languages.

• What does this entail?

  Examine the way in which languages are designed and implemented.
Class Objectives.

• What are we going to do in this class?
  Compare and contrast different programming languages.

• What does this entail?
  Examine the way in which languages are designed and implemented.
Why do this?

1. For the **fun** of it!

2. Understanding the basic principles makes it **easier to learn new languages**.

3. Sometimes you need **different features of different languages**, and if you don’t know about other languages how can you use them?

4. **More effectively utilize** the languages you already know.
Why do this?

1. For the **fun** of it!

2. Understanding the basic principles makes it easier to learn new languages.

3. Sometimes you need different features of different languages, and if you don't know about other languages how can you use them?

4. More effectively utilize the languages you already know.

   For example, if you need "fine-grained" control over system memory, then you **C++** would be a better choice than **Java**. However, if your memory leaks are a big concern, then **Java** is a better choice than **C++**.
A very very very brief history of languages.

- In the beginning, ENIAC (Electronic Numerical Integrator and Computer) programmers used **patch cords**.

- This gave them the raw power to compute trig tables.
Machine and Assembly Languages.

• The next major revolution was **machine language**, which is just binary (or hexadecimal).

• Very quickly people realized that humans cannot write error free programs using just zeroes and ones without going insane.

• Hence, came **assembly language**, which uses human readable abbreviations to stand for machine code.
Assembly language (example)

Start:
lea A, a0
lea B, a1
lea C, a2
clr.w d0
clr.w d1
clr.w d2
add.w #5, d1
add.w #6, d2
move.w d1, (a0)
move.w d2, (a1)
add.w (a0), d0
add.w (a1), d0
move.w d0, (a2)
jsr decout
jsr newline
jsr stop

data
A: dc.w 1
B: dc.w 1
C: dc.w 1
Higher level languages

• Eventually, people realized that more complex programs are very difficult to write at the level of assembly language.

• So, eventually came higher level languages.

```java
class Test {
    public static void main(String args[]) {
        int A, B, C;
        A=5;
        B=6;
        C=A+B;
        System.out.print(C);
    }
}
```
Declarative and Imperative programming

• There are two types of programming languages: declarative and imperative.
  • Declarative languages focus on what the computer should do.
  • Imperative languages focus on how the computer should do something.
Quicksort sorts an array by **recursively** sorting “sub-arrays” as less than or greater than **pivot values**.

X Pivot

Y Less then

Z Greater then
Quicksort in Haskell

\[
\text{qsort } [] = [] \\
\text{qsort } (x:xs) = \text{qsort } \text{lt}_x ++ [x] ++ \text{qsort } \text{ge}_x \\
\text{where} \\
\text{lt}_x = [y \mid y \leftarrow xs, y < x] \\
\text{ge}_x = [y \mid y \leftarrow xs, y \geq x]
\]
Quicksort in Haskell

If input is **empty** return **empty**.

```haskell
qsort [] = []
qsort (x:xs) = qsort lt_x ++ [x] ++ qsort ge_x
  where
    lt_x = [y | y <- xs, y < x]
    ge_x = [y | y <- xs, y >= x]
```

If input is empty return empty.
Otherwise, return a list with all the values **less than x** both “`qsort`”ed and **before x** and all values **greater than x** both “`qsort`”ed and **after x**.

```haskell
qsort [] = []
qsort (x:xs) = qsort lt_x ++ [x] ++ qsort ge_x
  where
    lt_x = [y | y <- xs, y < x]
    ge_x = [y | y <- xs, y >= x]
```
This junk defines \( \text{lt}_x \) as all values \textbf{less than} \( x \), and \( \text{ge}_x \) as all values \textbf{greater than or equal to} \( x \).
Quicksort in C

```c
qsort( a, lo, hi ) int a[], hi, lo;
{
    int h, w, p, t;
    if (lo < hi) {
        w = lo;
        h = hi;
        p = a[hi];
        do {
            while ((w < h) && (a[w] <= p))
                w = w+1;
            while ((h > w) && (a[h] >= p))
                h = h-1;
            if (w < h) {
                t = a[w];
                a[w] = a[h];
                a[h] = t;
            }
        } while (w < h);
        t = a[w];
        a[w] = a[hi];
        a[hi] = t;
        qsort( a, lo, w-1 );
        qsort( a, w+1, hi );
    }
}
```
Find the **first element larger than the pivot value** and the **last element smaller than the pivot value**.

```c
qsort( a, lo, hi ) int a[], hi, lo;{
    int h, w, p, t;
    if (lo < hi) {
        w = lo;
        h = hi;
        p = a[hi];
        do {
            while ((w < h) && (a[w] <= p))
                w = w+1;
            while ((h > w) && (a[h] >= p))
                h = h-1;
        } while (w < h);
        t = a[w];
        a[w] = a[hi];
        a[hi] = t;
        qsort( a, lo, w-1 );
        qsort( a, w+1, hi );
    }
}
```
If these values are on the “wrong side” of the pivot, swap them.
The University of North Carolina at Chapel Hill

Quicksort in C

```c
qsort( a, lo, hi ) int a[], hi, lo;
int h, w, p, t;
if (lo < hi) {
    w = lo;
    h = hi;
    p = a[hi];
    do {
        while ((w < h) && (a[w] <= p))
            w = w+1;
        while ((h > w) && (a[h] >= p))
            h = h-1;
        if (w < h) {
            t = a[w];
            a[w] = a[h];
            a[h] = t;
        }
    } while (w < h);
    t = a[w];
    a[w] = a[hi];
    a[hi] = t;
    qsort( a, lo, w-1 );
    qsort( a, w+1, hi );
}
```

Swap the **smallest value greater than or equal to the pivot** with the **pivot**, which is at the end of the list.
Finally, recurse on the two sides.
Quicksort in C

Quicksort in C

Notice how much more complex this program is in C (an imperative language) than Haskell (a declarative language).
Quicksort in C

```c
qsort( a, lo, hi ) int a[], hi, lo;
{ int h, w, p, t;
  if (lo < hi) {
    w = lo;
    h = hi;
    p = a[hi];
    do {
      while ((w < h) && (a[w] <= p))
        w = w+1;
      while ((h > w) && (a[h] >= p))
        h = h-1;
    if (w < h) {
      t = a[w];
      a[w] = a[h];
      a[h] = t;
    }
  } while (w < h);
  t = a[w];
  a[w] = a[hi];
  a[hi] = t;
  qsort( a, lo, w-1 );
  qsort( a, w+1, hi );
}
```

However, without a very good compiler, the quicksort in C will likely run faster than in Haskell!
Types of Languages

Declarative
- Functional
  e.g., Haskell & Lisp
- Dataflow
  e.g., Id & Val
- Logic
  e.g., Prolog

Imperative
- Von Neumann
  e.g., Fortran, Basic, & C
- Object-Oriented
  e.g., C++ & Java
- Scripting
  e.g., Perl
Functional languages are based on functions and recursion.

Declarative
- Functional
  - e.g., Haskell & Lisp
- Dataflow
  - e.g., Id & Val
- Logic
  - e.g., Prolog

Imperative
- Von Neumann
  - e.g., Fortran, Basic, & C
- Object-Oriented
  - e.g., C++ & Java
- Scripting
  - e.g., Perl
Dataflow languages focus on the flow of information between nodes.

Declarative:
- Functional: e.g., Haskell & Lisp
- Dataflow: e.g., Id & Val
- Logic: e.g., Prolog

Imperative:
- Von Neumann: e.g., Fortran, Basic, & C
- Object-Oriented: e.g., C++ & Java
- Scripting: e.g., Perl
Logic languages model programs as a series of logical statements.
Von Neumann languages allow for computation by focusing on manipulating data elements.

- **Declarative**
  - Functional
    - e.g., Haskell & Lisp
  - Dataflow
    - e.g., Id & Val
  - Logic
    - e.g., Prolog

- **Imperative**
  - Von Neumann
    - e.g., Fortran, Basic, & C
  - Object-Oriented
    - e.g., C++ & Java
  - Scripting
    - e.g., Perl
Object-oriented languages allow for computation by modeling principles as a series of semi-independent “objects”.  

- **Declarative**
  - Functional: e.g., Haskell & Lisp
  - Dataflow: e.g., Id & Val
  - Logic: e.g., Prolog

- **Imperative**
  - Von Neumann: e.g., Fortran, Basic, & C
  - Object-Oriented: e.g., C++ & Java
  - Scripting: e.g., Perl
**Scripting** languages are a subset of von Neumann languages and are serve as “glue” between more robust languages in order to facilitate rapid development.
Course Topics

• Tentative List:
  • Compilation & Interpretation
  • Syntax Specification & Analysis
  • Names, Binding, & Scope
  • Control Flow
  • Data Types
  • Subroutines & Control Abstraction
  • Concurrency
  • Code Improvement
  • Data Abstraction & Object Orientation
  • Scripting Languages: Perl, Python, Ruby, etc..
  • Functional Languages: ML, Lisp/Scheme, Haskell, etc…
  • Logic Languages: Prolog
  • and more…