Programming Languages

General Purpose language

HTML

Constants

Variables

Data Types

Functions

Operators (+ - ! < >)

Control Structures

Conditional (if-then)

Loops (For, While)

Interpreted

Compiled

High level

Low Level

Matlab

C

C++

Java

Syntax

Families

Why you Care

Any tool you choose, or have written for you, will be in some particular language.

Languages all have strengths and weaknesses which affect their speed, compatibility with the OS, tweakability, and cost.

You should have a rough idea of how language choices affect you.

General Purpose Languages

General purpose languages (of which there are hundreds or thousands) can be used to accomplish the same set of tasks.

Some, however, are better than others for a given task.

By analogy, you could write your thesis in Word, Excel, Notepad, or Powerpoint,--all of them handle text input.

But Word is best suited to the particular task.

Other "Languages"

- HTML is NOT a programming language.
- It is a markup language; designed to provide structure to a web document.
- That is, it "marks" bold, italic, paragraphs, tables, links etc.
- But
  - you can't extend it with new functions
  - it can't do computations

Programming languages are really like languages

They are grouped in **families**.

They each have their own **syntax**.

They have *noun-like* objects: **constants** and **variables** that fall into different classes (**data types**).

And they have *verb-like* actions that operate on the nouns: procedures, operators, methods, functions and control structures.

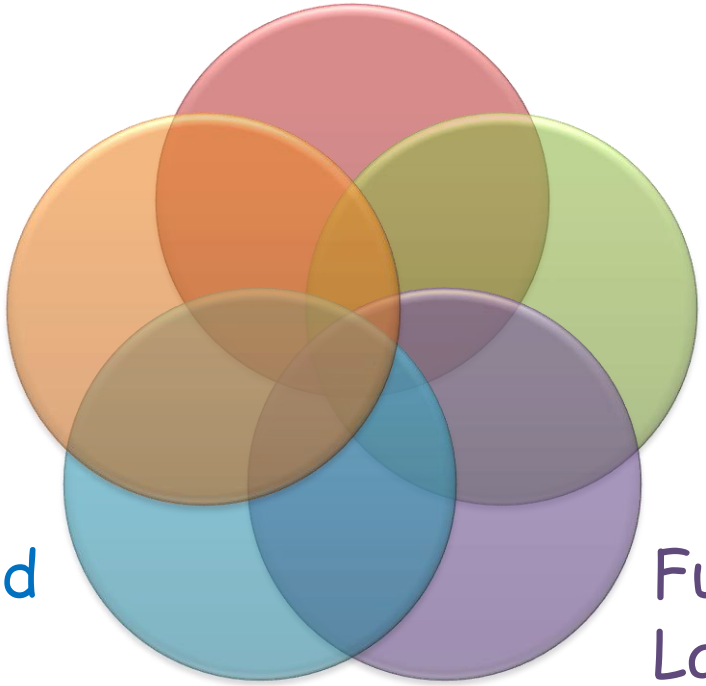
Sample Language Families

Array Languages:

- e.g., APL, Matlab, IDL, R

Procedural Languages

- Basic, C, Java, Matlab



Assembly Languages

- BAL, GAS, MIPS

Object Oriented Languages

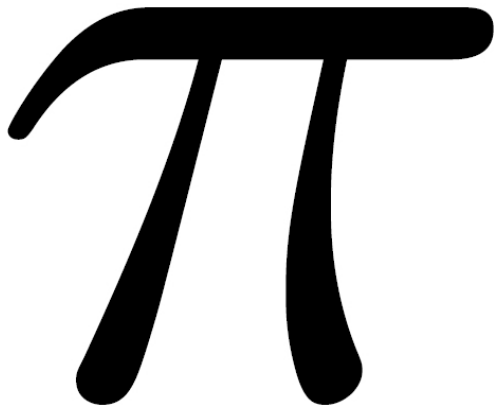
- C++, Java, Groovy

Functional Languages

- Lisp, Mathematica, R

Constants

- Constants have a single, unchanging value.
  - e.g., pi, 6, "Fred"



Variables

- Variables are like containers. You can vary the values that they hold.
  - e.g., for each image file in the directory, stick the filename in the variable **x**, zip **x** up, log **x**'s size, and then get the next filename, put it in **x**, repeat……

Data Types

- Constants and variables have types that constrain their behaviors, e.g.:
  - integers (1,2,3,4),
  - floating point numbers (6.939, 4.556),
  - characters (x,y,3, $),
  - booleans (true, false),
  - strings ("you can't do that", 'hello world')
- Only certain operations are allowed for each data type:
  - You can't take the square root of a string.

Operations on Data Types

- Function: operation that returns a value
  - strlen('abc') ➔ 3
  - sqrt(4) ➔ 2
  - plus(sqrt(4), sqrt(9)) ➔ 5
- Operator: a function represented by a symbol: `+ - / > < ! ~ = ==`
  - 'foobar' - 'bar' ➔ foo
  - "foo" + "bar" ➔ foobar
  - 'x' * 4 ➔ xxxx
  - (5 * 3) == 8 ➔ false

Control Structures

- Conditional statements
  - **if** the image has no size, **then** delete it.
- **For** loops
  - **For** each image in this directory, smooth the image.
- **While** loops
  - **While** image1 and image2 are different sizes, erode image2 again
- Others: languages vary in the control structures they provide.

Syntax differs between languages

bash:

```
if [ x –gt 10 ]
then
  my_variable = 4
else
  my_variable = 88
fi
```

C or Java:

```
int my_variable;
if (x > 10)
  my_variable = 4;
else
  my_variable = 88;
```

Matlab:

```
if  x > 10
  my_variable = 4;
else
  my_variable = 88;
end
```

Groovy:

```
def my_variable = (x > 10) ? 4 : 88
```

Lisp:

```
(setq my_variable (if (> x 10) 4 88))
```

But, more importantly,
different programming
languages
have different strengths and
weaknesses.

Interpreted vs Compiled

- **Interpreted** languages (like shell scripts) are executed line by line with no translation from source code.
  - easy to modify, slow to run, portable
- **Compiled** languages (like C) are translated from source code into machine code before they are run.
  - produces a fast program, but not human readable once translated.
  - May be a pain to translate and is platform specific.

High Level vs Low Level

High level languages

- are more user friendly,
- and more like natural language
- hide details of CPU operations and memory access

Low level languages

- tend to lack abstract structures
- specify exactly how to handle hardware resources like memory addresses

Matlab

- Interpreted
  - easy to work with, modify and share across different platforms,
  - but slow
- Good at matrix math
- Terrible at string manipulation
- Very slow for-loops
- Used for image and signal manipulation because images and signals are easily represented as matrices
- Expensive commercial product (like IDL)

C and C++

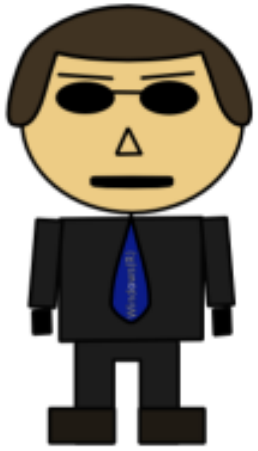
- Compiled specifically for each platform.
- Hence, programs run fast.
- Complex to write and debug.
- Compilation process is often complex.
- Missing more modern language features
- Prone to memory leaks
- Used mostly for systems and scientific programming.

Java

- Portable, high level, modern language
- Compiled, but the compiled program runs on different platforms via the JVM (Java Virtual Machine).
- With the JVM on your machine, you can run any compiled Java program.
- Complex, like C and C++
- Often used for web programming

Which Language will you Use?

C	C++	Matlab & IDL	Java
low-level, complex fast	mid-level, complex, fast	well managed tools at a price; slow	C++ done right (fast, high level, complex)
platform specific compilation	platform specific compilation	interpreted	compiled but portable
OS, low level tools	Scientific applications	Scientific applications	Portable and web applications



Pick the right one for the job!

General Purpose language

HTML

Constants

Variables

Data Types

Functions

Operators (+ - ! < >)

Control Structures

Conditional (if-then)

Loops (For, While)

Interpreted

Compiled

High level

Low Level

Matlab

C

C++

Java

Syntax

Families