HOW DOES COMPUTER PROGRAMMING WORK?

MAGIC.
Computer Components

Software

User Programs

Operating System

Hardware
What are Programs?

- Programs provide instructions for computers
- Similar to giving directions to a person who is trying to get from point A to point B. A program may say... if starting at location A, go north 3 blocks, then go East 2 blocks.
What are Programs?

Input

Processing

Output
High/Low Level Programming Languages

• Low level programming languages are languages which have instruction sets that are specific to the computer hardware being run on. Programmers need to know how the hardware works to use it.

• High level programming languages provide a layer of abstraction that allows for the programmer to only have to learn a hardware independent language to write software.
Low Level Languages

- **Pros**
  - Can be more optimized
  - Usually smaller executables
- **Cons**
  - Platform dependent
  - Slower development
  - Harder to understand
  - Easier to introduce bugs
- **When to use it:** Need extra optimization, Need small executables (embedded systems)
- **Example:** Assembly (different for each OS/Architecture)
High Level Languages

- **Pros**
  - Faster development (libraries/etc.)
  - Easier to understand (abstraction)
  - Can be platform independent

- **Cons**
  - Usually larger executables
  - Abstraction layer usually adds overhead for processing resulting in slower executables

- **When to use it:** Need portability, need faster turn-around time for development, when programmer time is more valuable than processing time

- **Examples:** Java, C, C++, C#, Javascript, PHP, Perl, Python, Lisp, Scheme, R, etc.
Assembly Example (Low Level)

mov ax,cs
mov ds,ax
mov ah,9
mov dx, offset Hello
int 21h
xor ax,ax
int 21h
Hello:
   db "Hello World!",13,10,"$"
Python Example (High Level)

print(“hello World!”)
Java Example (High Level)

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

Anatomy of a Program

Traditional Language

Source Code

Compiler

Compiled Code

Linker

Executable

Interpreted Language

Source Code

Interpreter

Byte Code

Virtual Machine

Execution
Bytecode vs. Executable

- **Bytecode**
  - Cross platform
  - Allows for replacement of small components without recompiling entire programs
  - Generally slower performance

- **Executable**
  - Runs on one platform
  - Programs generally compile down to larger executables
  - Generally faster performance
Algorithms
Algorithms

- A series of instructions that solve a problem
Algorithms

• A series of instructions that solve a problem. Example, sorting a list of numbers:

repeat:
    swapped = false
    for i = 1 to length(numbers) – 1:
        /* if this pair is out of order */
        if numbers[i-1] > numbers[i] then
            /* swap and remember change */
            swap( numbers[i-1], numbers[i] )
            swapped = true
        end if
    end for
until no swaps occur
Pseudo-Code

- The expression of programming logic in a language independent nature.
- Good for design phase of coding
- Example: the last slide!
Bugs
Bugs

- All programmers run into bugs... lots and lots of bugs. Bugs are errors in a program.
- Bugs come in two different varieties:
Bugs

• All programmers run into bugs... lots and lots of bugs. Bugs are errors in a program.

• Bugs come in two different varieties:
  • Compile time errors: (syntax/type/etc.) The rules of the language have been violated. Examples: improper spacing, missing colon, etc.
Bugs

• All programmers run into bugs... lots and lots of bugs. Bugs are errors in a program.

• Bugs come in three different varieties:
  
  • Compile time errors: (syntax/type/etc.) The rules of the language have been violated. Examples: improper spacing, missing colon, etc.
  
  • Runtime errors: Errors during execution
    
    – Logic errors: A problem that is caused by a flawed algorithm or set of instructions used to solve a problem. Example: If you use a less than sign where you should have used a greater than sign
    
    – Could cause program to crash
Writing a Program

1. Design
2. Write Code
3. Fix Compile Time Errors
4. Run Program
Basic Programming Concepts

- Keywords
- Statements
- Operators
- Data Types
- Variables
- Assignment
Keywords

- Keywords - reserved words that have special meaning in a particular programming language. These words can not be used for any other purpose (e.g. variable names). Java keywords are:

  abstract, continue, for, new, switch, assert, default, goto, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum, instanceof, return, transient, catch, extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp**, volatile, const*, float, native, super, while
Statements

- Instructions telling the computer what to do
Basic Data Types

- Integers (Real numbers): short (16 bits), int (32 bits), long (64 bits)
- Single text character: char
- 32bit decimal numbers: float
- 64bit (more precise) decimal numbers: double
- True/False (similar to a bit): boolean
- 8 bits: byte
- Strings (are actually objects): store a series of characters
Operators

- Different data types have different operators
- Operators can be part of a statement
- Operators act on the operands around them
- Unary operators – take 1 argument
  - e.g. setting a number to a negative value: -3
- Binary operators – take 2 arguments
  - e.g. arithmetic operators: 1 - 3
- Ternary operators – takes 3 arguments
  // if x < y set largest to y, else set it to x
  largest = x < y ? y : x;
### Operators on Numbers

In order of increasing precedence, with respect to dividers (i.e. + and – have the same precedence, but execute after *,/,%).

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Additive operator (also used by strings)</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction operator</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication operator</td>
</tr>
<tr>
<td>/</td>
<td>Division operator</td>
</tr>
<tr>
<td>%</td>
<td>Remainder operator</td>
</tr>
<tr>
<td>( )</td>
<td>Dictate order of operations</td>
</tr>
</tbody>
</table>
Comments

- Provides:
  - Documentation
  - Clarifying what specific code is doing
  - Make code easy for the author or other programmers to understand

- Line comments start with the `//` characters
  
  ```
  // this is a comment
  ```

- Block comments start with `/*` and end with `*/`
  
  ```
  /* this is a block comment
   * comment, it spans multiple lines */
  ```
Variables

- Variables are labels used by programmers for storage of data
- Variable name in java:
  - Can be long
  - Are case sensitive (Alpha != alpha)
  - Can contain letters, numbers, and underscores (_)
  - Must not start with a number
  - Can not be a keyword
Variables

• The variable's data type (boolean, int, float, double, char, etc.) must be declared when created!

• The variable's data type will never change (i.e. it will always store that type of data)
Variables Names (Examples)

thisVariableNameIsValid = "valid naming conventions"
this_variable_name_is_valid = "also valid"
names = "names are valid"
Names = "not the same as names"
numbers_between_0_and_9 = "another valid"
0_9_can_not_start_a_name = "invalid"
if = "keywords are not allowed"
special_chars_!_allowed = "invalid"
Variable Assignment Statements

// constant PI
static final float PI = 3.14159265;

// variables radius/area_circle
float radius = 5;
float area_circle = PI * radius * radius;
Constants

- A constant is a stored value that doesn't change
- Used for things that will remain constant throughout the program. (e.g. pi, conversions between metric and standard units, etc)
- When representing data that doesn't change it's often a good idea to use constants
- Generally constants are in all CAPITAL letters
Data Output

- Use: `System.out.println()`
- API documentation for 'System' is here: [http://docs.oracle.com/javase/7/docs/api/java/lang/System.html](http://docs.oracle.com/javase/7/docs/api/java/lang/System.html)
- API documentation for 'println()' is here: [http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html](http://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html)
Data Output

• For numbers with strings:

    String year = “2014”
    System.out.print(“The year is: ” + year)
Data Input

• For strings:

Scanner reader = new Scanner(System.in);
System.out.println("Enter input: ");
String input=reader.nextLine();

• For numbers:

Scanner reader = new Scanner(System.in);
System.out.println("Enter an integer");
int input=reader.nextInt();
Natural Languages

- Syntax: punctuation and spacing
- Semantics: The meaning of the words
• Mixing numbers with strings:

String year = "2014";

// Not what we want (string concatenation):
System.out.println("Next year is: " + (1 + year);

// Convert string to int then add:
int nextYear = Integer.parseInt(year)++;
System.out.println("Next year is: " + nextYear);
Type Conversion

- In certain circumstances your data types may need to be converted to other data types (e.g. converting a string into an integer or vice versa).

- String → int
  
  ```java
  foo = "31337'';
  int bar = Integer.parseInt(foo);
  ```

- int → String
  
  ```java
  Integer.toString(bar)
  ```
Type Conversion (more)

- Variables are declared as a specific type
- Variables are that type for the life of that variable
- Variable types cannot be changed once declared

```java
String year = "2014";
// errors (because year is forever a String):
year = Integer.parseInt(year);

// instead, create an int variable to store it in:
int iYear = Integer.parseInt(year);
```
Programming Languages

- Syntax: punctuation (e.g. parentheses, colons, spacing)

- Semantics: The meaning of the words
  - What does “System.out.println()” do?
Coding Style

- Just as in natural languages the style of coding matters.
- Style determines the level of readability, maintainability, and efficiency.
- Several things make up a coding style:
  - Formatting, Naming schemes, Comments, and more!