Programs are:
• **Source code** written in a text editor,
• Following the syntax of a language,
• Specifying both memory locations (variables) and instructions (statements),
• That must be translated into a form the computer can actually use (often binary instructions directly executable by the CPU).

Errors
• Syntax Errors
  – Violations of the rules of the language
• Run-Time Errors (Bugs)
  – Computations giving the wrong results
  – Computations halting the program (unchecked divide-by-zero, for example)
• Both require editing the source text of the program, retranslating it, and trying again.

Languages
• Early compiled languages (FORTRAN, COBOL, ALGOL, PL/I) from the 1950s and 1960s.
• Later compiled languages (Pascal, C, C++, Ada) from the 1970s and 1980s.
• Early interpreted languages (BASIC, LISP, APL) from the 1960s.
• Later interpreted languages (Python, JavaScript, Perl, many scripting languages for Web servers)
• Modern languages compiled to a “generic” computer model, then interpreted by a virtual machine (Java)

Flowcharts
• Provide a visual, non-language-specific way of describing a program,
• Used to be how programmers designed programs in the first place,
• Are a good teaching tool to illustrate how programs work.

Translators
• Assemblers
  – Translate very low-level statements into binary instructions (1→1), creating stand-alone .EXE files.
• Compilers
  – Translate high-level statements into many binary instructions (1→many), creating stand-alone .EXE files.
• Interpreters
  – Translates and executes each statement as it is encountered, requiring translator to run programs.

1/19/2020
Example: Factorial

- The factorial of an integer $N$ is the product of all integers from 1 up through $N$.
- $N$ factorial is written as $N!$.
- $N! = 1 \times 2 \times 3 \times \ldots \times N$ (iterative definition)
- $N! = N \times (N-1)!$ (recursive definition)
- $0! = 1$ (makes recursion work)
- $5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

Flowcharts

- Here’s the flowchart version of the factorial program:

Tracing Flowcharts

- Put your finger on the START box,
- Follow the flow-arrows,
- When you enter a box do what it says,
- Update the variables appropriately,
- Don’t take your finger off until you hit STOP.
What Does This Give Us?

- By following a flowchart, we see how computers execute their programs,
- We also see how detailed programs must be to accomplish any task,
- But computers do each step extremely fast (on the order of a few nanoseconds).

Programs may be written in Different Ways:

- Some are shorter
- Some are faster
- Some use less memory
- Some use bizarre techniques
- Some are easier to teach
- Some are easier to debug
- Some languages are easier than others
The Factorial Program in Python 3

```python
N = int(input("Enter Number: "))
F = 1
I = 1
while (I <= N):
    F = F * I
    I = I + 1
print (F)
```

Here's the same program in JavaScript (embedded in HTML Web Page)

```javascript
<!--
N = parseFloat(window.prompt("Enter Number: "));
F = 1;
I = 1;
while (I <= N) {
    F = F * I;
    I = I + 1;
}
document.writeln (F);
//-->
</SCRIPT>
```

Here's the same program in Pascal

```pascal
Program Factorial;
Var N,F,I : Integer;
Begin
    Write ('Enter Number: ') ;
    Readln(N) ;
    F := 1 ;
    I := 1 ;
    While (I <= N) Do
    Begin
        F := F * I ;
        I := I + 1 ;
    End;
    Writeln (F) ;
End.
```

Here's the same program in BASIC

```basic
10 INPUT N
20 LET F = 1
30 LET I = 1
40 IF I > N THEN 80
50 LET F = F * I
60 LET I = I + 1
70 GOTO 40
80 PRINT F
90 END
```

Here's the same program in 8088 Assembly Language

```assembly
MOV AX,1 ; F=1
MOV BX,5 ; N=5
MOV CX,1 ; I=1
TopLoop: CMP CX,BX ; Test I:N
         JG EndLoop ; Jump if >
         MUL CX ; F=F*I
         ADD CX,1 ; I=I+1
         JMP TopLoop ; Jump back
EndLoop: CALL PRINT ;
```

Languages (1/3)

- Python:
  - Interpreted,
  - Dynamically Typed,
  - One statement per line,
  - Whitespace (indentation) determines lexical scope.
- JavaScript (not Java):
  - Interpreted (typically by Web browser),
  - Dynamically Typed,
  - Statements terminated by semicolons (;),
  - Curly braces ({})) determine lexical scope.
Languages (2/3)

- Pascal:
  - Compiled,
  - Statically Typed,
  - Statements separated by semicolons (;)
  - Keywords (Begin-End) determine lexical scope.
- BASIC (as originally implemented):
  - Interpreted,
  - Statically Typed (suffixes carry type: A, A$, A%),
  - One statement per line,
  - What lexical scope?

Languages (3/3)

- Java (not JavaScript):
  - Compiled to intermediate form interpreted by JVM,
  - Statically Typed,
  - Statements terminated by semicolons (;)
  - Curly braces ({}) determine lexical scope.
- Assembly Language:
  - Assembled (for particular machine architecture),
  - Instructions carry type (ADD vs. FADD),
  - One statement per line,
  - What lexical scope?

The Factorial Program in Python 3
Again

\[
\begin{align*}
N &= \text{int(input("Enter Number: "))} \\
F &= 1 \\
I &= 1 \\
\text{while } (I \leq N): & \\
\quad & F = F \times I \\
\quad & I = I + 1 \\
\text{print } (F)
\end{align*}
\]

Here's a more efficient way

\[
\begin{align*}
N &= \text{int(input("Enter Number: "))} \\
F &= 1 \\
\text{for } I \text{ in range}(1,N+1): & \\
\quad & F = F \times I \\
\text{print } (F)
\end{align*}
\]

Here's a radically different way:

\[
\begin{align*}
def \text{Factorial}(N): & \\
\quad & \text{if } (N \leq 1): \text{ return } 1 \\
\quad & \text{else return } N*\text{Factorial}(N-1)
\end{align*}
\]

On the 105 Final Exam...

- I will provide a flowchart of roughly this complexity (but not the same program),
- I will provide boxes for each of the variables,
- You will trace the flowchart and determine the final results.
- I will NOT ask you to draw a flowchart.