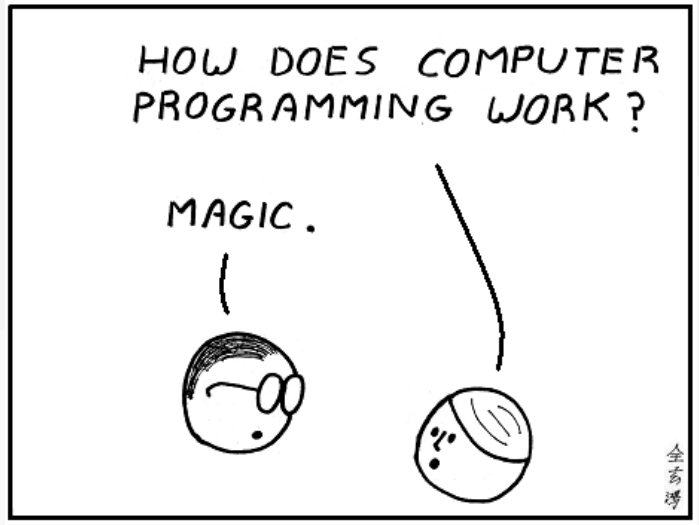
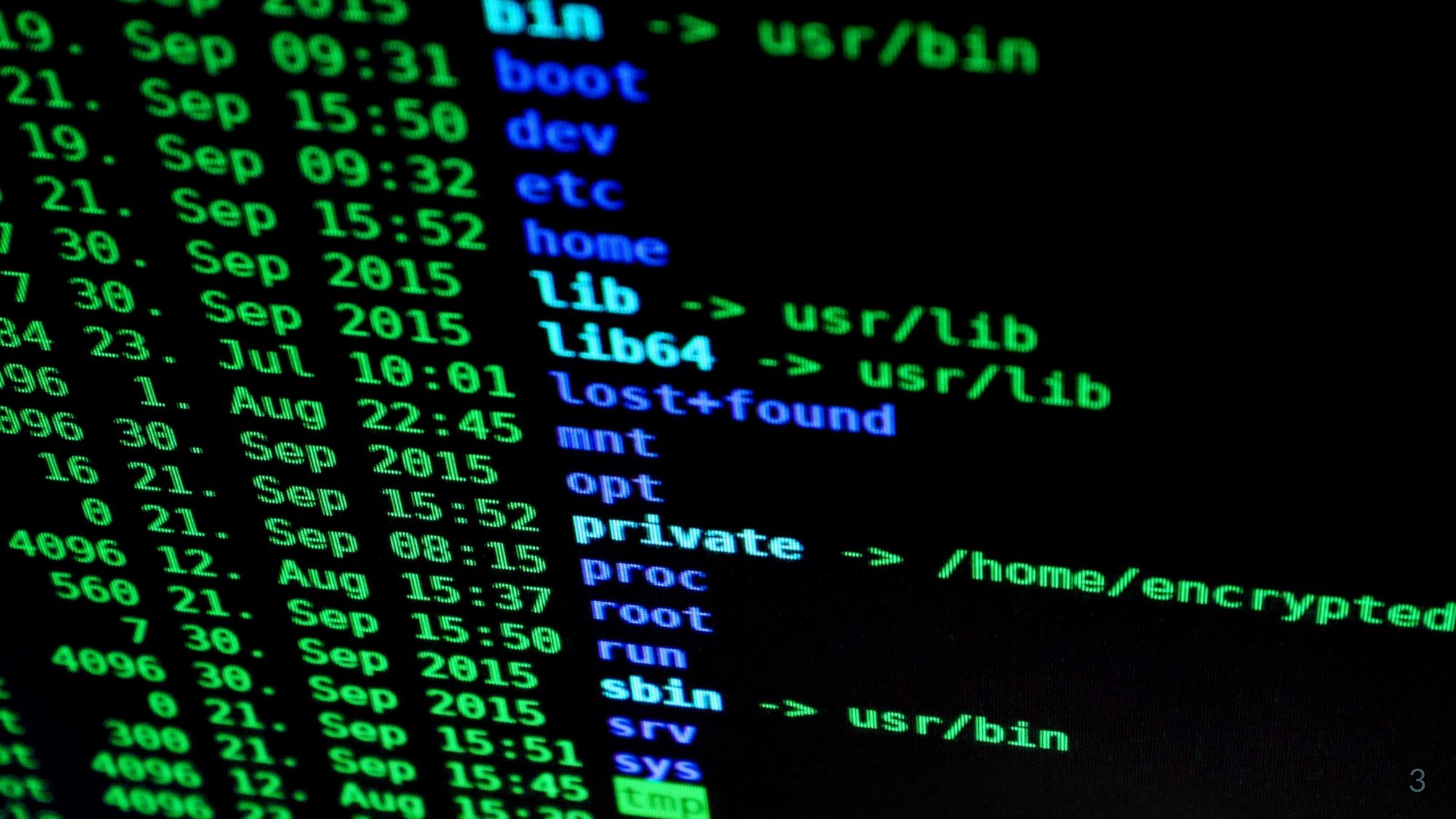
Introduction to Computer Programming

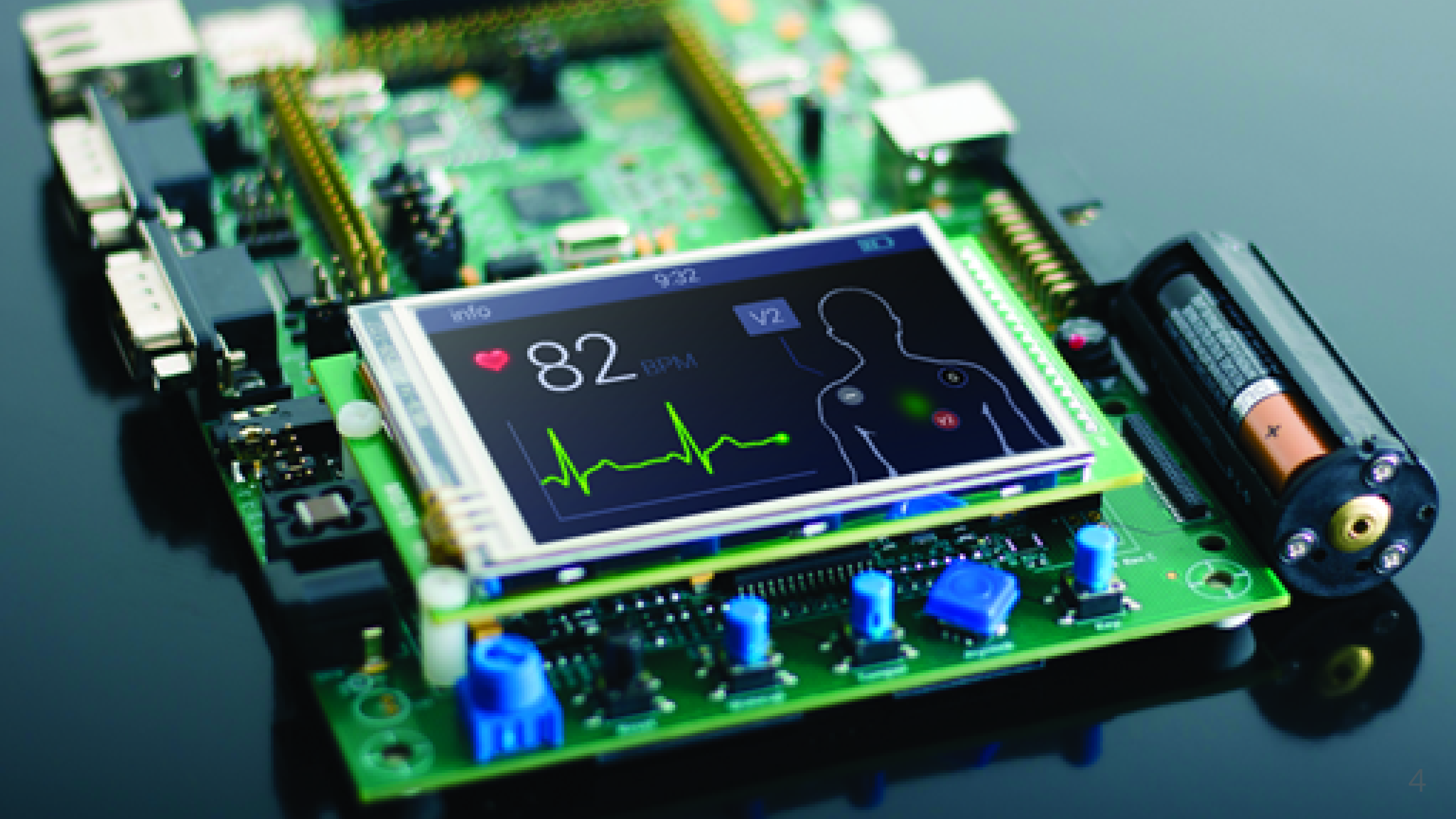
Chapter 01

Introduction




tetris

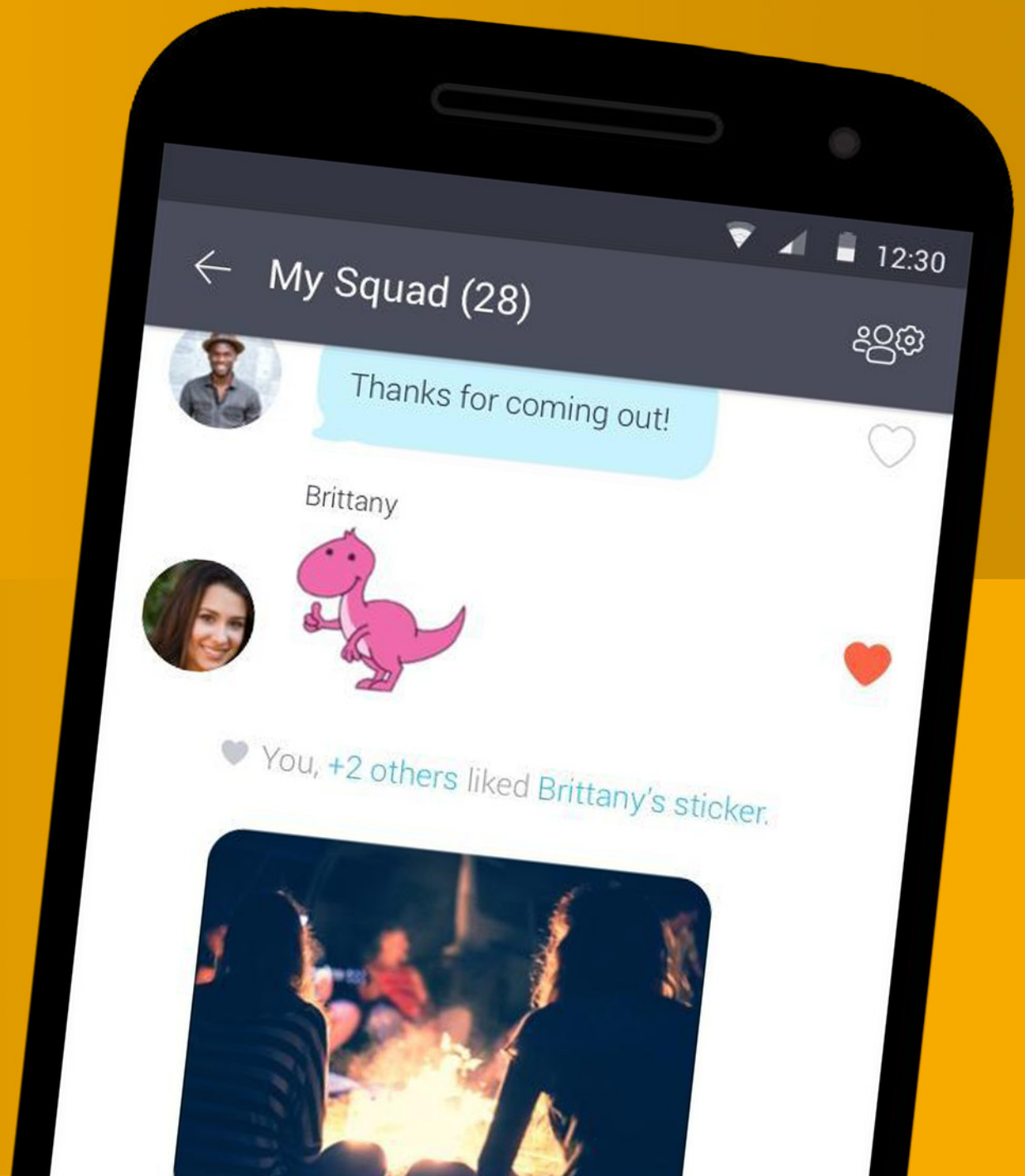

bin -> usr/bin
boot
dev
etc
home
lib -> usr/lib
lib64 -> usr/lib
lost+found
mnt
opt
private -> /home/encrypted
proc
root
run
sbin -> usr/bin
srv
sys
tmp


82
BPM

Web Application


Windows 11
11:11 AM

best. Android Apps

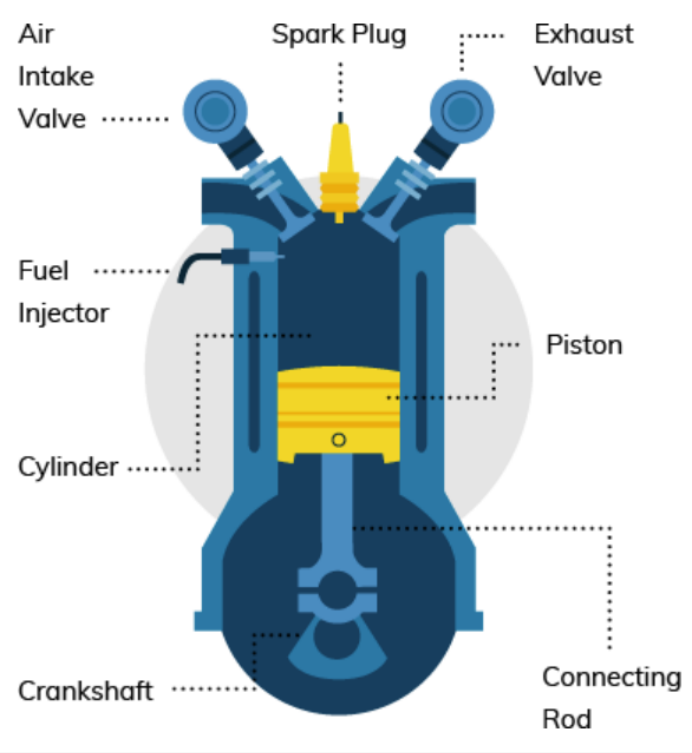


Common Factor?

Why Everyone Should Learn to Code

Introduction

- Most people don't actually know how a computer works
- They can interact with a computer
- Impossible to know how everything works
  - Do you know internal working of car?
  - Does not mean we can't drive



Writing Software

- As a programmer you will need to know a bit how a computer works
- Writing software =
  - **describing processes and procedures**
  - **authoring of algorithms**
  - **developing lists of instructions**
  - = source code
  - = instructions that manipulate different types of data

Some Definitions

- “ **proc-ess / Noun:** A series of actions or steps taken to achieve an end. ”
- “ **pro-ce-dure / Noun:** A series of actions conducted in a certain order. ”
- “ **al-go-rithm / Noun:** An ordered set of steps to solve a problem. ”

Clarity of Expression

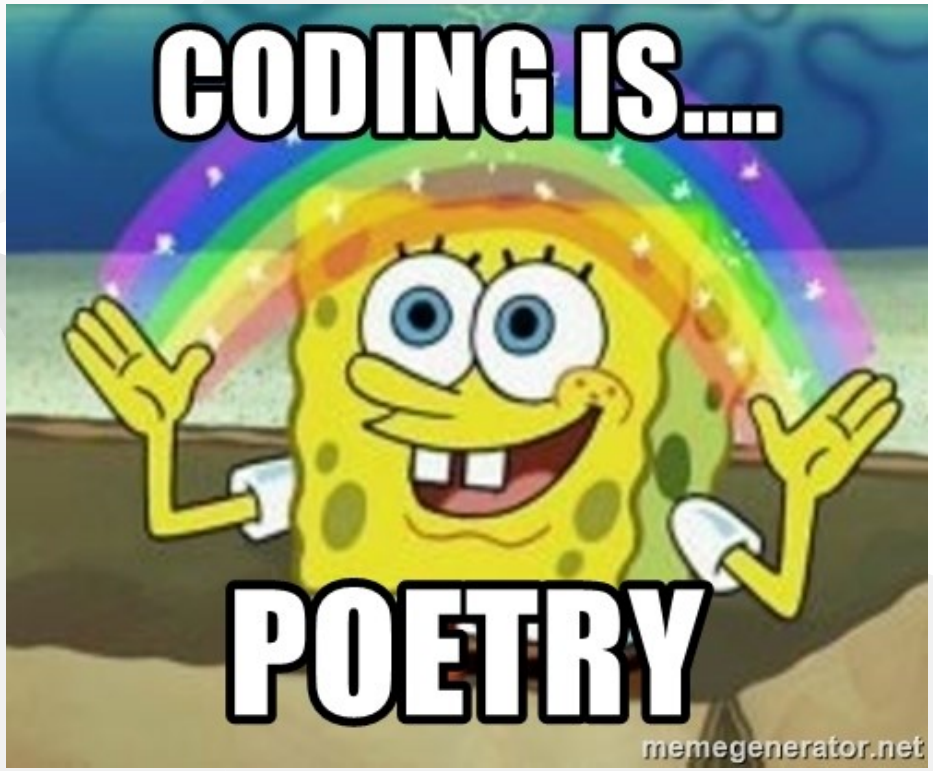
- Learning to programming is valuable
  - Even if you don't make a career out of it
  - Will help you to learn the importance of **clarity of expression**
- Why ?
  - **A computer is very dumb, but obedient**

Teaching

“ It has often been said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until after teaching it to a computer, i.e., express it as an algorithm.

(Donald Knuth, in "American Mathematical Monthly," 81) ”

Fun Fun Fun



- But, most of all, **it can be lots of fun**!
- Computer = your own little entity you get to boss around all day to do all kinds of neat stuff for you.

An Application

- A computer is a tool for solving problems
- An application is a **sequence of instructions** that tell a computer how to do a certain task.
- When a computer follows the instructions in a program, it is said **it executes the program**.

A Spoken Language

- Before we start programming
- Use a language such as the English language to d**escribe how to do something as a series of steps**
  - Making a ham-and-cheese grilled sandwich
  - Washing a car
  - Doing laundry
  - ...

A Spoken Language

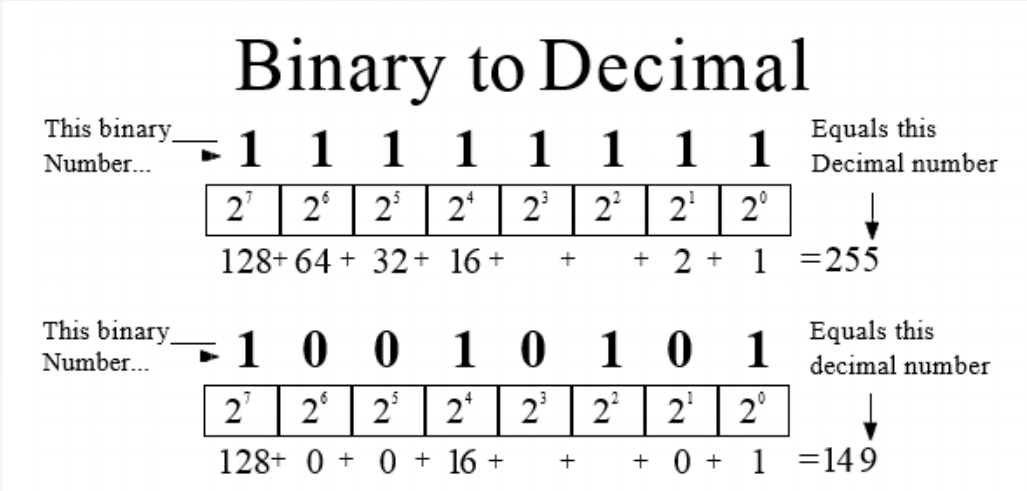
- Was that easy ?
  - Did you remember all the steps?
- Useful exercise
  - Can become very complex
- Computers are just not ready for it yet
- Heck, most humans aren't even ready for it yet.

Don't Blame the Computer

- Don't blame the Computer, blame the programmer
- Instead ask yourself:
  - Did I tell the computer how to do the job correctly?
  - Did I forget something?
  - Did I misinterpret the problem
  - Do I have the solution wrong?

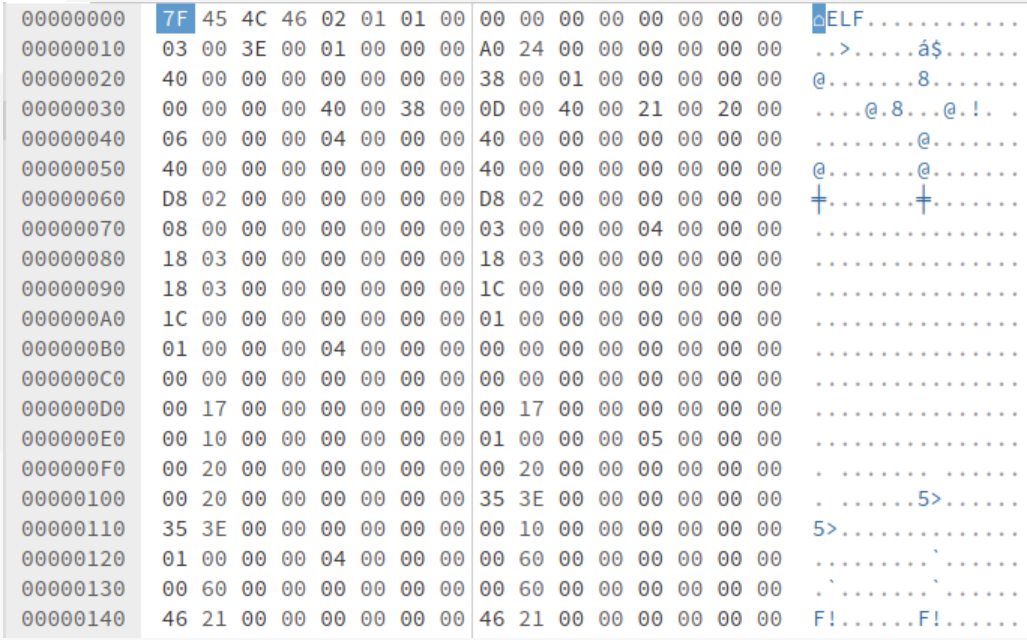
The Binary Language

- Computer don't understand recipes written on paper
- Computers are machines
  - a collection of electronic switches
  - 1 represents "on" and 0 represents "off".
- Everything that a computer does is implemented in this most basic of all numbering systems - **binary**.



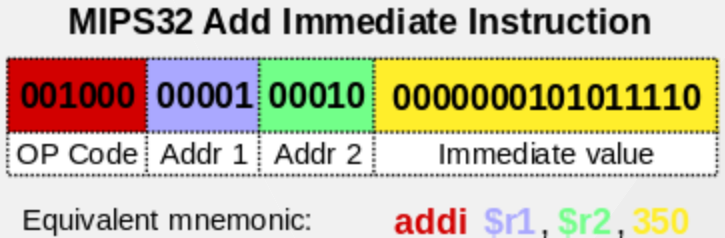
Machine Code

- Want to really tell a computer what to do ?
- You'd have to talk to it in binary, giving it coded sequences of 1s and 0s
  - Tell it which instructions to execute - **machine code**
- In practice, we use a programming language.



A Programming Language

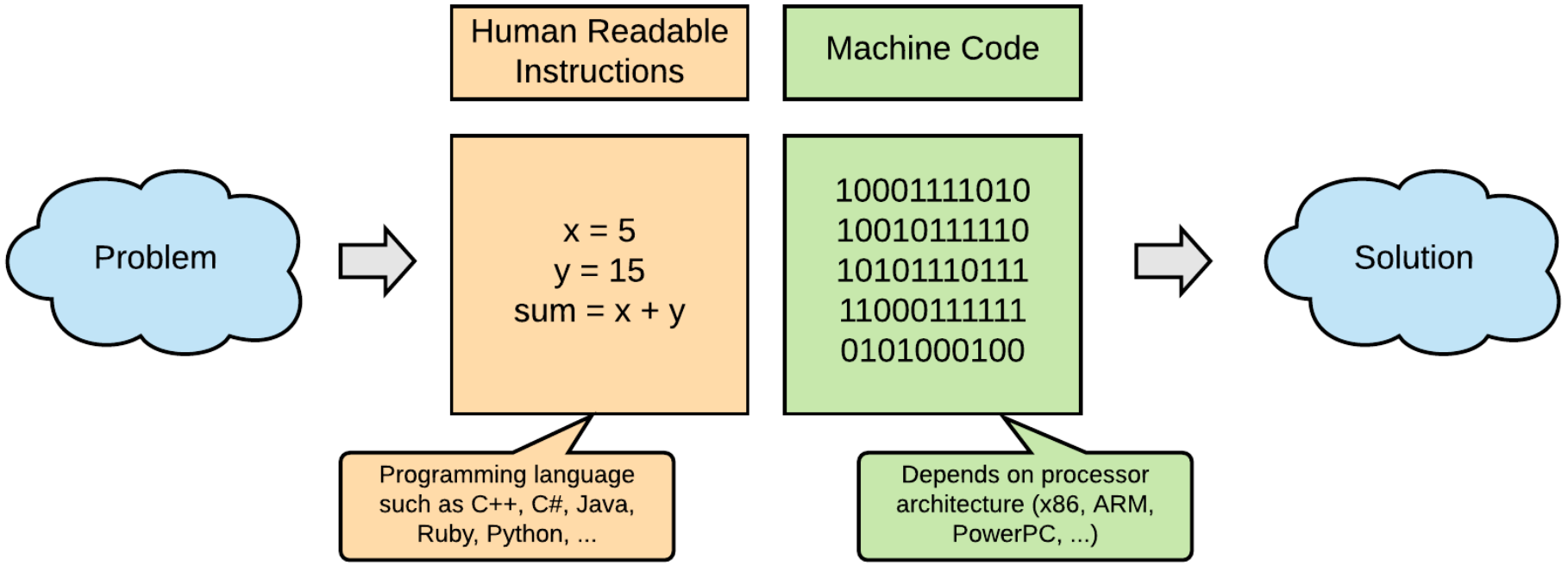
- A language developed to express programs
- All computers have native programming language = **machine code**
  - Tell the processor what to do
  - Impractical for us humans
  - Unique to a particular computer architecture (x86, ARM, PowerPC, ...)
  - Processor instruction set



Abstraction is Key

- Abstraction is the process of hiding complex things behind a simpler interface
- **Higher level programming languages** do exactly this
  - BASIC
  - Java, C#, C++
  - ...
- Easier for us to understand
- Less dependent on actual hardware

Abstraction is Key



Abstraction is Key

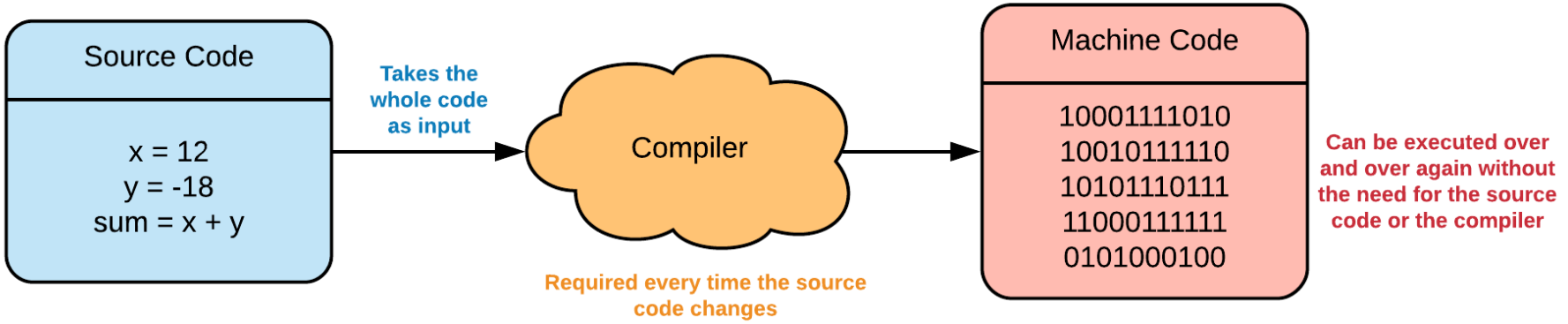
- These higher-level languages are said to abstract away the complexity of the underlying system.
- Higher level programming languages still need to be **translated into machine code**.
  - Compiled
  - Interpreted
  - (hybrid)

Compilation

- Tool = compiler
- Translation of higher language in **architecture dependant machine code**
- Input = full source code
- Output = executable binary file that is permanently stored

Compilation

- Analogy: book in different languages
- Compiler
  - Transforms source code that was written in a specific programming language into another
  - Not just machine code

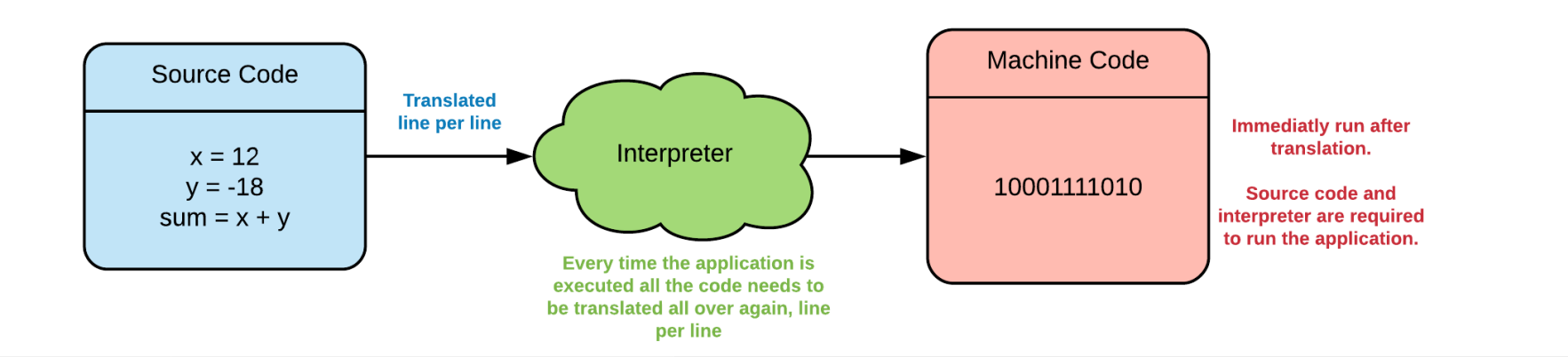


Interpretation

- Tool = interpreter
- Translation of higher language in **architecture dependant machine code** @ runtime
- Input = partial source code
- Output = machine code to be run at that moment

Interpretation

- Analogy: human interpreter
- Interpreter
  - At no point is a complete, discrete, machine code version of the program produced



Compile or Interpret

- Not always your choice
- Depends on the programming language / tool you are using
- Many are hybrid forms these days
- General
  - Compiled programs are faster to run but slower to develop
- Architecture dependency

Transpile

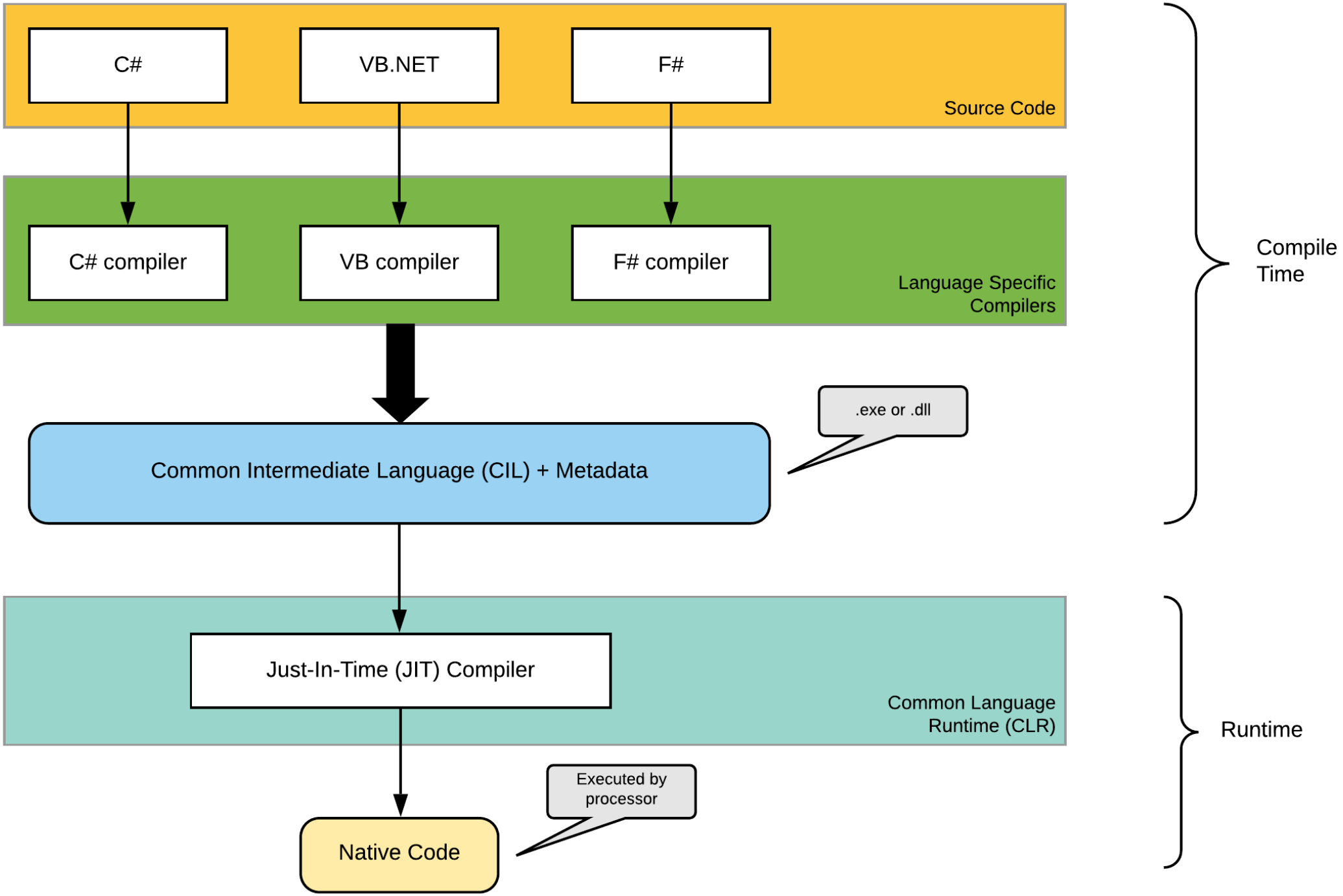
- Transpiler = Translate compiler
- Taking source code written in one language and transforming into another language that has a similar level of abstraction.
  - Output still has to go through another compiler or interpreter to be able to run on a machine.
- Some examples of transpilers are:
  - `tsc` or TypeScript compiler, transpiles TypeScript into JavaScript
  - `babel`, Transpiles ES6+ code to ES5 (ES6 and ES5 are different versions or generations of the JavaScript language)

Let's See a Demo

- Compiled: C++
- Interpreted: JavaScript
- Transpiled: TypeScript

What About C#

- @Compile Time
  - Code is compiled to **Common Intermediate Language (CIL)**
  - Language specific C# compiler
  - Result is executable binary: `.exe` (or `.dll` in case of a library)
- @Runtime
  - Binary can be run on system with .NET runtime installed
  - The Just-In-Time (JIT) compiler takes CIL code as input and **transforms it into the processor specific machine code**


C#
VB.NET
F#
Source Code
C# compiler
VB compiler
F# compiler
Language Specific Compilers
Common Intermediate Language (CIL) + Metadata
.exe or .dll
Just-In-Time (JIT) Compiler
Common Language Runtime (CLR)
Native Code
Executed by processor
Compile Time
Runtime

Levels of Programming Languages

- Low level programming languages
  - Closer to machine code
- High level programming languages
  - Closer to natural language

Assembly Language

- Most basic level
- Direct translation of the binary instructions
- Each assembly language instruction directly relates to one instruction in machine code
- So each processor architecture has its own instruction set with accompanying assembly language

An assembly example

```
LUI  R1, #1
LUI  R2, #2
DADD R3, R1, R2
```

- Calculation `1 + 2 = 3`
- First two lines load the numbers "1" and "2" into the computer's memory
- Third instruction tells the computer to add the values together and store the result

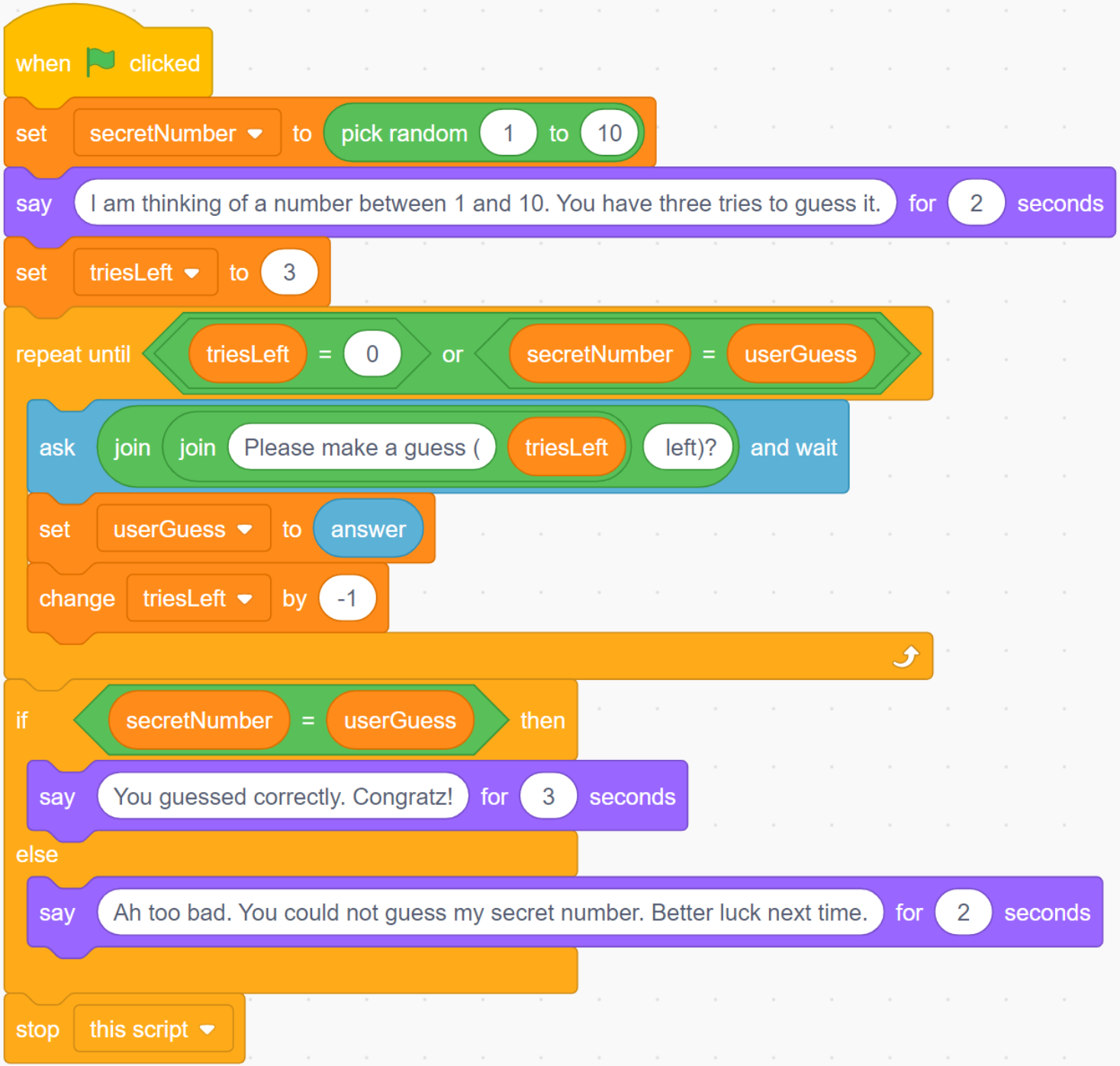
High Level Languages

- Assembly language is quite dissimilar to natural languages
  - Ultimate flexibility and performance, at the expense of complexity and development time.
- Higher level languages get closer to natural languages
  - More efficient to express
  - Look more like natural language with mathematical operations thrown in

```
int x = 1 + 2;
```

Graphical Programming Languages

- More than 15 years ago, Scratch was invented
  - By Mitchel Resnick and friends at MIT
  - New approach to teaching computer programming
  - Graphical programming language
  - Programs are constructed by connecting blocks
- Fun way to get started in programming
  - Not a way to create professional applications


when clicked
set secretNumber to pick random 1 to 10
say I am thinking of a number between 1 and 10. You have three tries to guess it. for 2 seconds
set triesLeft to 3
repeat until triesLeft = 0 or secretNumber = userGuess
ask join join Please make a guess (triesLeft left)? and wait
set userGuess to answer
change triesLeft by -1
if secretNumber = userGuess then
say You guessed correctly. Congratz! for 3 seconds
else
say Ah too bad. You could not guess my secret number. Better luck next time. for 2 seconds
stop this script

Applications

- Applications come in many different kinds and flavors.
  - **service in the background**: ex. webserver
  - **in a terminal**: ex. git
  - **graphical application**: ex. Visual Studio
  - **inside a browser**: ex. Scratch editor

Console Applications

- Designed to be used via a **text-only computer interface**, such as
  - a text terminal, the command line interface of some operating systems (Unix, DOS, etc.)
  - or the text-based interface included with most Graphical User Interface (GUI) operating systems, such as the Win32 console in Microsoft Windows, the Terminal in Mac OS X, and xterm in Unix.
  - **Interaction happens using keyboard**

```
posh~git ~ paho.mqtt.cpp [master]
C:\mqtt_libraries\paho.mqtt.c [master ≡ +14 ~1 -0 !]> cd ..
C:\mqtt_libraries> cd .\paho.mqtt.cpp\
C:\mqtt_libraries\paho.mqtt.cpp [master ≡]> cmake -G"MinGW Makefiles" -DPAHO_MQTT_C_PATH="..\paho.mqtt.c" .
-- The CXX compiler identification is GNU 4.8.1
-- Check for working CXX compiler: C:/mqtt_libraries/mingw-posix-sjlj-rev5/mingw64/bin/g++.exe
-- Check for working CXX compiler: C:/mqtt_libraries/mingw-posix-sjlj-rev5/mingw64/bin/g++.exe -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/mqtt_libraries/paho.mqtt.cpp
C:\mqtt_libraries\paho.mqtt.cpp [master ≡ +11 ~1 -0 !]> make
Scanning dependencies of target OBJS
[  7%] Building CXX object src/CMakeFiles/OBJS.dir/async_client.cpp.obj
[ 15%] Building CXX object src/CMakeFiles/OBJS.dir/client.cpp.obj
[ 23%] Building CXX object src/CMakeFiles/OBJS.dir/disconnect_options.cpp.obj
[ 30%] Building CXX object src/CMakeFiles/OBJS.dir/iclient_persistence.cpp.obj
[ 38%] Building CXX object src/CMakeFiles/OBJS.dir/message.cpp.obj
[ 46%] Building CXX object src/CMakeFiles/OBJS.dir/response_options.cpp.obj
[ 53%] Building CXX object src/CMakeFiles/OBJS.dir/ssl_options.cpp.obj
[ 61%] Building CXX object src/CMakeFiles/OBJS.dir/string_collection.cpp.obj
[ 69%] Building CXX object src/CMakeFiles/OBJS.dir/token.cpp.obj
[ 76%] Building CXX object src/CMakeFiles/OBJS.dir/topic.cpp.obj
[ 84%] Building CXX object src/CMakeFiles/OBJS.dir/connect_options.cpp.obj
[ 92%] Building CXX object src/CMakeFiles/OBJS.dir/will_options.cpp.obj
[ 92%] Built target OBJS
Scanning dependencies of target paho-mqttpp3-static
[100%] Linking CXX static library libpaho-mqttpp3.a
[100%] Built target paho-mqttpp3-static
C:\mqtt_libraries\paho.mqtt.cpp [master ≡ +12 ~1 -0 !]>
```

Console Applications

- Use of console applications has greatly diminished, but not disappeared
- Some users simply prefer console based applications
- Some organizations still rely on existing console applications to handle key data processing tasks.

Console Applications

- Another huge advantage of working with console applications compared to a GUI application is the **ability to automate certain tasks**.
  - Can be chained
  - Easily used in automation scripts
- Still best te develop when learning to program

GUI Applications

- GUI or Graphical User Interface applications
- Require interaction with mouse and keyboard
- Not so easy to automate
- Not best choice to learn to program

Preview a_programming_language.md - gitbook - Visual Studio Code

File Edit Selection View Go Debug Tasks Help

previously translated instructions. In interpreted languages, every time the program is run the computer also needs to translate each of the instructions. This translation causes a delay, slowing the execution of the program. On the other hand, interpreted languages are often written in a smaller time frame, because whole program does not need to be compiled each time a new feature is implemented.

The table below gives an overview of the most important differences between a compiler and an interpreter.

	A Compiler	An Interpreter
Input	Takes an entire program as its input	Takes a single line of code as its input
Output	An executable	None
Execution Speed	Faster	Slower as each instruction needs to be translated to machine code
Development Speed	Slower as it needs to be compiled every time	Faster
Errors	Often harder to find as code might be optimized	Easier as execution happens line by line

What About Java

Java implementations typically use a two-step compilation process. Java source code is compiled down to bytecode by the Java compiler. The bytecode is then executed by a Java Virtual Machine (JVM). Modern JVMs use a technique called Just-in-Time (JIT) compilation to compile the bytecode to native instructions understood by hardware CPU on the fly at runtime.

Some implementations of JVM may choose to interpret the bytecode instead of JIT compiling it to machine code,

```
Compiled programs often run faster because the computer only needs to execute
the previously translated instructions. In interpreted languages, every time
the program is run the computer also needs to translate each of the
instructions. This translation causes a delay, slowing the execution of the
program. On the other hand, interpreted languages are often written in a
smaller time frame, because whole program does not need to be compiled each
time a new feature is implemented.
26
27  The table below gives an overview of the most important differences between a
    compiler and an interpreter.
28
29  |   | A Compiler | An Interpreter |
30  | --- | --- | --- |
31  | Input | Takes an entire program as its input | Takes a single line of code as
    its input |
32  | Output | An executable | None |
33  | Execution Speed | Faster | Slower as each instruction needs to be translated
    to machine code |
34  | Development Speed | Slower as it needs to be compiled every time | Faster |
35  | Errors | Often harder to find as code might be optimized | Easier as
    execution happens line by line |
36
37  #### What About Java
38
39  Java implementations typically use a two-step compilation process. Java source
    code is compiled down to bytecode by the Java compiler. The bytecode is then
    executed by a Java Virtual Machine (JVM). Modern JVMs use a technique called
    Just-in-Time (JIT) compilation to compile the bytecode to native instructions
    understood by hardware CPU on the fly at runtime.
40
41  Some implementations of JVM may choose to interpret the bytecode instead of JIT
```

```
Loading personal and system profiles took 892ms.
D:\OneDrive\Vives\courses\object_oriented_programming_with_java\gitbook [dev +16 ~4 -1 !]> git status
On branch dev
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   01_introduction_to_computer_programming/a_programming_language.md
        modified:   01_introduction_to_computer_programming/an_application.md
        modified:   01_introduction_to_computer_programming/console_gui.md
        deleted:    01_introduction_to_computer_programming/img/Bash_screenshot.png
        modified:   01_introduction_to_computer_programming/readme.md

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        01_introduction_to_computer_programming/img/a_modern_terminal.png
```

How to Become a Programmer

- Understand the problem
  - Define an appropriate solution
  - Express that solution in a computer programming language
- Practice is essential
- Don't be afraid to make mistakes
- Learn to work in team

> An expert is a man who has made all the mistakes which can be made, in a narrow field. - *Niels Bohr*