ComIT

Introduction to Programming

Purpose of Programming

- In order to being programming, we need to have an understanding of what it is that we are trying to achieve.
- What is programming for? What is the goal of a program?

Purpose of Programming

- In order to being programming, we need to have an understanding of what it is that we are trying to achieve.
- What is programming for? What is the goal of a program?
- We program to solve problems.
  - This begs the question, “what kind of problems?”

Solving a Problem

- Lets go through the process of solving a problem
- Pretend you are traveling with a friend and you get a flat tire.
- For some reason, you cannot change the tire yourself, but you know how. How would you instruct your friend to change the tire?

Group Exercise

Example – Qualified Friend

- If your friend was a mechanic, the steps might be very basic.
  - Your friend knows what they are doing, they may only need a few instructions, like where the spare tire is, and they may be able to do the rest.
- Example:
  1. Remove spare from trunk
  2. Get jack from under hood.
  3. Replace tire

Example - Inexperienced Friend

- If you are directing someone that knows nothing about changing tires, your instructions may be different.
- You will need to provide more detail, perhaps even a painful about of detail, before you are successful.

Context is Key

- The instructions you give are not universal, they depend on context.
- A key part of context is something we call an **interlocutor**
  - An interlocutor is an entity that takes part in a conversation and sometimes acts as a middleman
  - In order to be successful, we *need to speak their language*
- How is this relevant to programming computers?

Programming Computers

- Computers solve computational problems by moving and manipulating data.
- In order for a computer to do anything at all, it needs to be given instructions.
- So how do we send computers instructions?
  - What language does a computer speak?

Machine Language

- Computers only know very basic language rules, they work in a world of ones and zeros.
- We need to use machine language to talk to them
- We send instructions directly to the central processing unit (CPU)
  - Tasks like loading a word from memory, or jumping to a different instruction
- Different CPU architectures have different instruction sets that they understand.

Assembly Language

- This is a much more human readable rendition of a machine language.
- Does not use numeric values directly, they are replaced with mnemonic codes that are easier to remember and understand.
- It is a “low level language”, it provides little to no abstraction from a computer’s instructions.
- Do computers speak assembly?
  - Nope!
  - We must translate assembly into machine code with a program called an “assembler”
- So why don’t we use assembly language when programming?

Pitfalls of Assembly

- There are two main problems with Assembly languages:
  - They are difficult to work with
  - They are not portable.
- How can we solve this problem?

High Level Languages

- High Level Languages are those that provide more abstraction away from the details of the computer hardware.
- They do not concern themselves with “registers” and “memory addresses”.
- These languages are easier for people to understand and are faster to develop in
- They are also not machine or architecture dependent.
  - High level languages are converted into machine code by programs called “translators”

Language Translators

- There are two main families of translators:
  1. Compilers – Translate high level languages into low level languages or machine code
  2. Interpreter – Directly executes commands without having them compiled first
- Python is an interpreted language... Kind of.
  - Instructions are transformed into intermediary code, which is then interpreted.

Questions?