| Understanding and designing a programming language | *Xavier Rosado* | *January 20, 2021* |
|---|---|---|

- *Programming languages are the core tools of all software engineer/developers who build digital infrastructure.*
- *The variety of language between them can be daunting for beginner developers to get accustomed to; sometimes even one language can have large inconsistencies with it's own vocabulary and implementation.*
- *Many developers are not aware of languages internals, leaving the language as a "magical" black box.*
- *Big programming languages can leave holes in a beginners learning, as they take some features at face value and fail to comprehend their function*

- *Use a and LLVM to design a small, limited programming language.*
- *The language should have a concise and sensible syntax, as well as vocabulary.*
- *This would allow for developers to become more familiar with a languages internals, and later promote the learning of compilers/interpreters, as well as contribute to already existing ones to improve quality and performance.*

- *Languages grow over time, and maintaining standards across its various developments can be difficult, creating strange patterns and inconsistencies in itself.*
- *Many general-purpose languages also try to cover many use cases, creating incredibly large ecosystems where inexperienced developers can perceive it as larger than life, feeding into the idea of working on a language as increasingly difficult.*
- *Beginner developers are exposed to such a big ecosystem that they are unable to find a consistent writing style/standard to follow*

*Countermeasures*

- *Work on a small language, giving basic functionalities that can help developers limit the scope of design and later broaden the scope of the project.*
- *Build on existing stacks like LLVM to avoid time constraint related difficulties while still allowing development and flexibility.*
- *Focus on the language as a learning tool, following in the footsteps of older languages such as BASIC.*
- *Use a statically typed language to focus on correctness and types when compiling, giving better insight into the internals of languages.*

*Check/Evaluate*

*Check if the language is capable of being used in a learning environment while being concise. If the language doesn't meet basic requirements, evaluate if it's related to the tech stack or the limitedness of the language's design.*

*Act/Standardize*

- *Continue to build on the language, implementing modern language features (generics, lambdas, etc…) while not losing the intuitiveness of the core language.*
- *Make sure the language is able to be used on various systems and isn't limited to one OS/architecture.*