Angie Bui Language Evaluation Cheat Sheet	Efficiency	Regularity	Security	Extensibility
JavaScript	- Java is partially interpreted and therefore won't be as efficient at runtime as fully compiled languages. - Memory garbage collection can be inefficient - Java is fairly verbose requiring additional work my the programmer to perform actions	- Generally good - Inconsistently in the assignment of objects by reference and primitive data types by value - Inconsistency some of the operators (e.g. length) across types	- Reference based memory management reduces the likelihood of memory reference errors - Static typing reduces the possibility of runtime errors impacting code safety	- Easy to add new types (classes) - Java supports generics allowing creation of templates for new types
TypeScript	↑Programmer Efficiency ↑Compile ↓Runtime	follows a consistent set of rules and conventions. It has a syntax that is easy to read and write, and it includes features such as optional static typing and type inference that reduce the chances of introducing bugs into the code.	adds an extra layer of security to JavaScript by allowing developers to catch type-related errors early in the development process through optional static typing and strict null checks.	supports creating reusable code through modules and libraries. It integrates well with popular frameworks such as React and Angular, and its package manager, npm, simplifies the process of installing and managing external packages. also includes a powerful type system that makes it easy to create custom abstractions and interfaces.
Haskell	↓Programmer Efficiency ↑Compile	- Pure vs impure isn't very	↑ No pointers to memory ↑is statically	↑big libraries ↑lots of typing abilities

	↑Runtime	orthogonal - $ is a bit dodgy	typed	No macros
Kotlin	↑Programmer Efficiency ↓Runtime ↓Compile (slower than Java)	Kotlin adheres to a consistent set of rules and has a clear syntax. This makes it easy to read, write and maintain code. It also helps to reduce the risk of introducing errors into the code. Kotlin has features like type inference, extension functions, and smart casting, which help to reduce repetitive code and make the code more concise.	statically-typed language where the type of every variable is known at compile time. This early detection of type-related errors is beneficial for development. Kotlin also offers null safety features that prevent null pointer exceptions, making it a safe language. Furthermore, Kotlin has additional features like immutable variables, safe casting, and exception handling, which contribute to its security and safety.	permits developers to create their own abstractions and libraries. It enables developers to add new functionality to existing classes without modifying their source code through extension functions. Kotlin supports functional programming constructs such as lambdas, higher-order functions, and tail recursion that allow writing expressive and concise code.
Go	↑Programmer Efficiency ↑Compile ↑Runtime	consistent and concise syntax, making it easy to read, write, and maintain code. It has strict formatting conventions that are automatically enforced, leading to fewer errors or bugs in the code. Its syntax is	It has a built-in garbage collector that prevents memory leaks and buffer overflows, strict typing, and strict checking of type conversions to prevent errors. It also has built-in encryption libraries and supports secure	supports libraries and packages. It provides a standard library with common functions and a package management system that simplifies the installation and management of external packages.

		minimalist and practical.	coding practices.	Go's built-in Goroutines and channels support concurrent programming, which helps make applications efficient and scalable.
Rust	↓Programmer Efficiency ↑Runtime ↓Compile	regular language with a well-defined syntax and grammar. It enforces a strict formatting convention that is automatically checked by the compiler, which makes it easy to read, write, and maintain code. Rust's syntax is expressive and concise, with a focus on safety and performance.	has several features that help prevent common security vulnerabilities. Rust's ownership model and expressive type system help prevent memory-related errors and data races, while its standard library includes encryption libraries and support for secure coding practices.	supports creating reusable code through modules, crates, and libraries. Its package management system, Cargo, simplifies the process of installing and managing external packages, while Rust's macro system allows for compile-time metaprogramming, making it easy to extend the language itself. interoperability with other languages, especially C, makes it easy to integrate with existing codebases.