# Hoshie Lang

A data processing programming language

As the field of computer science expands to include ever more methodologies and specialities, from mathematical proofs to artificial intelligence, weather modelling to financial modelling, general purpose programming languages end up being the Swiss army knife of tools, ok at everything, but poor at any one specific job. One such speciality is data processing, covering everything from initial extract, transform and load workloads, to big data exploration and back to tabulation of complex results.

Further, general purpose languages offer many solutions to data processing, often encouraging the user to read an entire file into memory and processing it many times into new memory structures. This methodology if often good enough for “simple” tasks, but rarely scales well. On the other hand, efficient streaming or more modern generator paradigms can be harder to implement when the programmer is not familiar with them.

“Hoshie Lang” is a new programming language that specialises in processing data. It has the following goals:

1. Enable the programmer to succinctly define what they want to happen in their data process.
2. Provide the programmer with a built-in library of common algorithms and functions used to process and analyse data.
3. Provide a pleasant programming experience in an IDE.
4. Encapsulate the execution code generation in a way that can be extended and adapted in the future to ensure Hoshie Lang can be executed on a variety of platforms.
5. Remove the need for the programmer to be familiar with memory efficiency and streaming / generator techniques.

## Succinct Programming

The Hoshie Lang programming language is based around a “Pipeline” analogy. The programmer is encouraged to think of the data as “flowing” through a pipe (one row at a time), which in turn is made up of:

* **Activities**: A section in the pipe that alters the data in some way, typically:
  + Altering the content of a row.
  + Augmenting or Truncating the format of a row.
  + Removing specific rows.
  + Appending new rows.
* **Sensors**: A section in the pipe that observes the data as it passes that point in the pipeline, these are typically **reducers** which calculate scalar values based on the data it has seen up to that point in time.

The language is **declarative**, which places a natural emphasis on breaking down the entire process into a smaller set of independent **actions**, which in turn can then be used in specific data processing pipelines.

Similarly, **sensors** can be defined and placed along the pipeline to observe the state of process at any given time.

Rectangle = Actions  
Ellipses = Sensors

## Built-in Activities and Sensors

The language includes a set of predefined functions that are used to define the actions and sensors. For example, some common actions are:

* **Filter**: Allow a subset of rows to flow through the pipe.
* **Map**: Maps a specific row structure to a new row structure.
* **FirstN**: Only allow N rows to pass through.

And some common Sensors would be:

* **Count**: A simple row count.
* **Max**: The max value of a nominated field.
* **Mean**: The mean value of a nominated field.

## IDE

For languages to become popular and be usable, they need to have good integrated development environment support as this lowers the barrier of entry when learning the language and can increase the programmer’s productivity by an order of magnitude.

## Encapsulated Code Generation

One of the visions for Hoshie Lang is to be able to process data on a variety of platforms. For example:

* Small data could be processed within a web browser.
* Medium data could be processed within a middleware application, NodeJS, Python or Java.
* Large data could be processed in a distributed computing cluster like Hadoop or Spark.

By ensuring the code generation is well encapsulated adding new code generators can be a relatively easy exercise, preferably by domain experts in those target platforms. For this paper, the reference implementation is targeting JavaScript which can cover the small to medium data sizes above.

## Efficient Runtime

Having an efficient runtime will be an important part of Hoshie Lang. In the reference JavaScript implementation, a streaming based runtime is used, making heavy use of JavaScript generators. This effectively means that data rows are “pulled” from the end of the pipe, triggering the upstream actions to do “just enough” work for each row to be created.

For example, if the source dataset is 1 billion rows in length, but the processing pipeline contains a “FirstN” action (e.g., one which only lets the first 100 rows through), then the bulk of source dataset will never be processed.

## Conclusion

There are a lot of moving parts when creating a new programming language, all of which play a key part to the overall usability of the language.

There are also a lot of tools, libraries and IDEs available to assist in the task. This paper will take an in depth look at these and how they can be utilised to create a final product. Specifically:

* Language design.
* Compiler compilers, lexers and parsers.
* IDE integration and APIs
* Data processing and streaming techniques
* Documentation, unit test and build tools