Sparkground

Demo: igregory.ca/sparkground

A block-based digital playground for learning functional programming

Ian Gregory
CSCD94: Computer Science Project, Winter 2024
University of Toronto, Scarborough

Background

Many young students have their first interactions with computer programming using MIT's *Scratch*. In Scratch, users assemble interlocking *blocks*, representing actions, into sequences of steps that on-screen characters will follow.

Block-based programming creates a tactile digital space to experiment and play. Learners choose blocks directly from an organized collection, and the results of actions are immediately visible. Concerns of syntactic correctness fall away, while students focus instead on building semantic understanding with a tight feedback loop.

Like many programming languages, Scratch encourages *imperative programming*—describing a computational task as a series of instructions. While this is both prominent and practical, an alternative natural approach is *functional programming*—describing a computational task as a composition of mathematical functions (such as $f : A \to B$). Many functional languages were developed with teaching in mind, but they do not support block-based programming.

We have block-based programming.
We have functional teaching languages.

Sparkground is a fusion of these prior arts—a block-based digital playground for learning functional programming.

Principles

- Purely functional core language
- Tactile editing experience
- Rapid, informative feedback
- Optional static type checking as a guide for error diagnosis
- Visual data representation
- Learning with play

Future Work

- Comprehensive user testing with beginner programmers
- Type system extensions (user-defined structures, union types, bounded generics)
- Unit testing system
- Interactive debugging
- Visualization tools for data and algorithms

Implementation

Sparkground is implemented as a web application, available at igregory.ca/sparkground. Source code is linked from the Help menu.

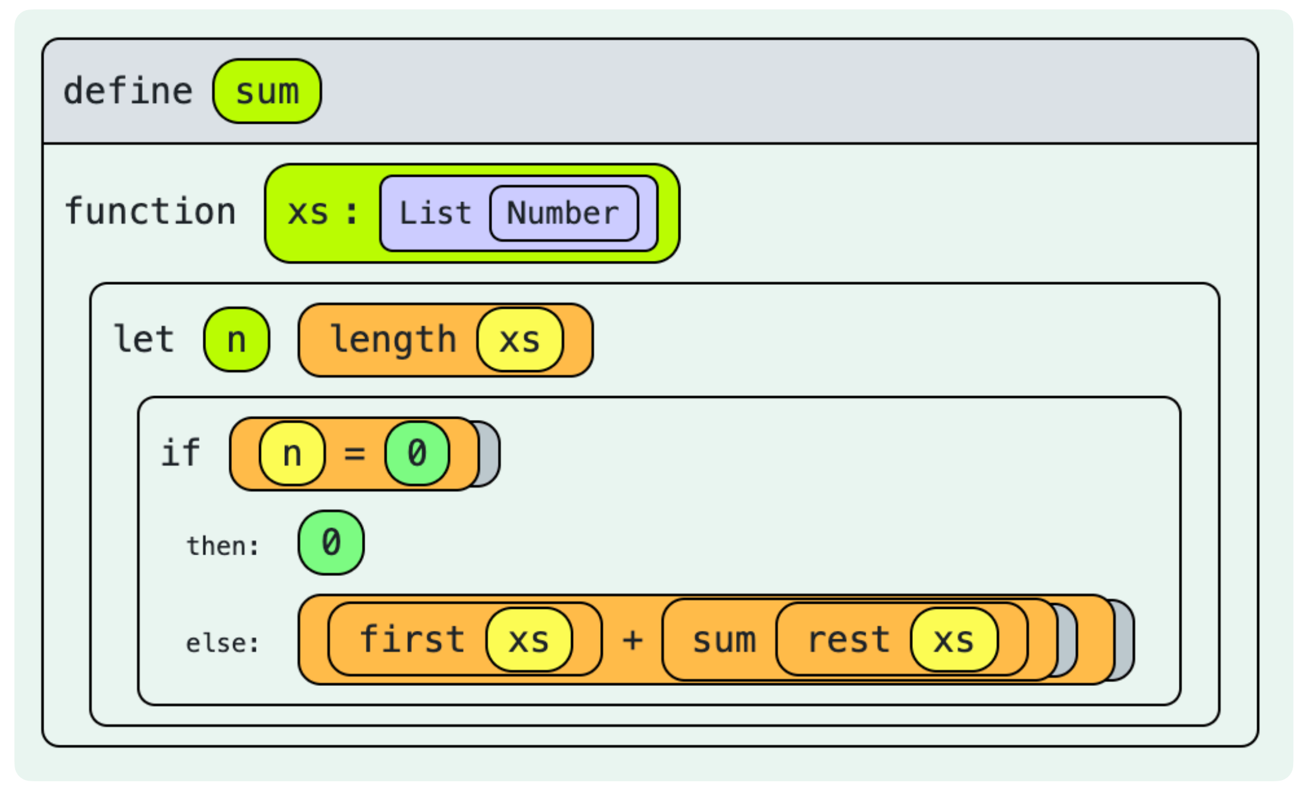
Drag-and-drop editing is inspired by Scratch, but with the functional tenet of *expressions, not statements*—every block results in a value, and can be used as input to another block (function).

The editor incorporates a gradual type system (optional static typing) to help diagnose errors, with support for subtyping and parametric polymorphism (generics). A *local type inference* scheme is employed, similar to that used by Java, Scala, Swift, and many other languages.

Example: Sum function

$$\mathrm{sum}(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } n = 0 \\ x_1 + \mathrm{sum}(x_2, \ldots, x_n) & \text{otherwise} \end{cases}$$

Representation in Sparkground



```
define sum
function xs : List Number
  let n length xs
    if n = 0
      then: 0
      else: first xs + sum rest xs
```