TinyExcel: Tiny spreadsheet system

Technical dimensions of spreadsheets

Tomas Petricek, Charles University

tomas@tomasp.net
@tomasp.net
https://tomasp.net
https://d3s.mff.cuni.cz/teaching/nprg077

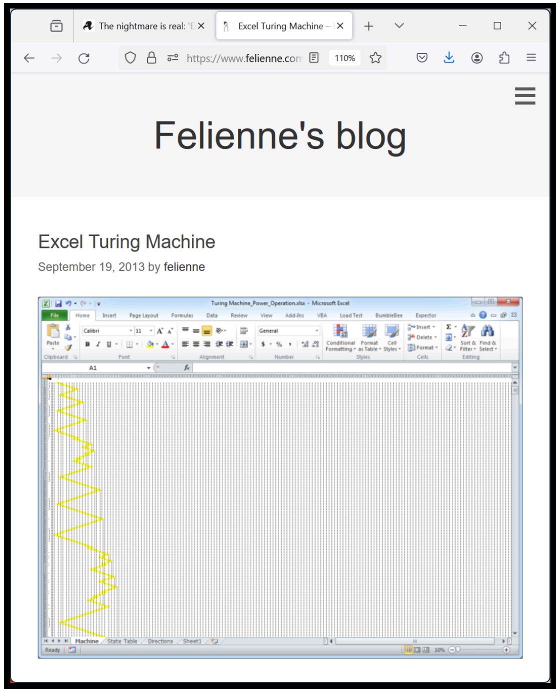
Is Excel real programming?

Charles University

Is Excel real programming?

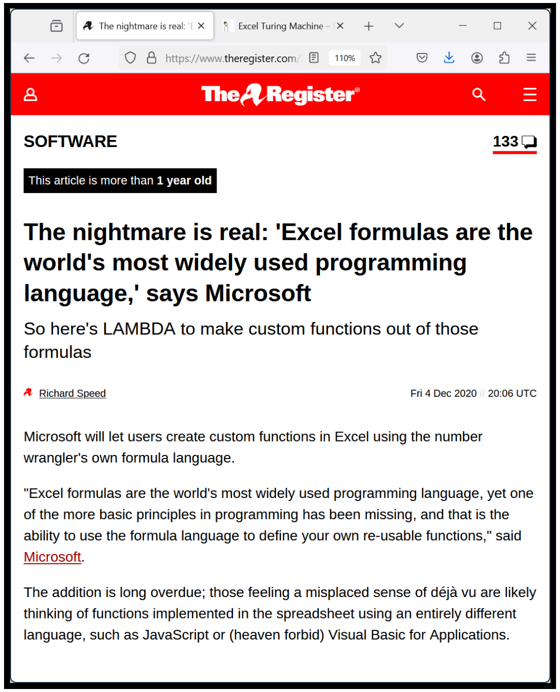
It is Turing-complete!

Encoded using "drag-down"



It is widely-used!

Simple, but can do a lot...


The Register

SOFTWARE 133

This article is more than **1 year old**

The nightmare is real: 'Excel formulas are the world's most widely used programming language,' says Microsoft

So here's LAMBDA to make custom functions out of those formulas

Richard Speed Fri 4 Dec 2020 // 20:06 UTC

Microsoft will let users create custom functions in Excel using the number wrangler's own formula language.

"Excel formulas are the world's most widely used programming language, yet one of the more basic principles in programming has been missing, and that is the ability to use the formula language to define your own re-usable functions," said Microsoft.

The addition is long overdue; those feeling a misplaced sense of déjà vu are likely thinking of functions implemented in the spreadsheet using an entirely different language, such as JavaScript or (heaven forbid) Visual Basic for Applications.

Charles University

TinyExcel

What makes spreadsheets interesting?

- Most accessible programming tools!
- Program in a two-dimensional space
- Edit and view in the same environment
- Automatic and live sheet recomputation

Technical Dimensions of Programming Systems

What matters about stateful interactive systems?

Jakubovic et al. (2023). See: https://tomasp.net/techdims

	→ Smalltalk	→ UNIX	→ Spreadsheets	→ Web platform	→ Notebooks	→ Haskell
→ Interaction	Integrated execution and editing mode, giving feedback at runtime. Abstractions constructed using objects are accessible via a browser.	Edit, build and execution modes with feedback in each step. Abstractions include files, memory and processes. Shell allows going from concrete to abstract.	Live update when editing. Formulas are always accessible. Abstraction by generalizing from concrete computation (drag down) or using macros.	Edit and refresh mode with state visible in DOM browser and live developer tools. Code abstractions are closed, but style abstractions more transparent.	Feedback and execution at cell level. Programmatic abstractions are possible, but manual approach by copying or modifying code is common.	Separate editing, compilation and execution modes with feedback at each level. Abstractions from first-principles (functions, type classes) are opaque during execution.
→ Notation	Primary source code notation with graphical structure editor for object structure. Secondary overlapping notations can be developed in-system. Small language.	Primary notation (the C language) with variety of secondary (file system, shell scripts), all edited via text editor. Admits concise but error-prone notations.	Complementing notations with graphical grid, formulas and macros, allowing gradually richer interactions. Different non-uniform notation at each level.	Diversity of text-based highly non-uniform notations (HTML, JavaScript, CSS) with limited structure editing for debugging (DOM).	Literate programming with code, text and outputs, embedded in a notebook as complementing notations. Document model where notebook is a list of cells.	Primary source code notation with secondary infrastructure notations, edited as text. Rich mostly explicit language with variety of extensions.
→ Conceptual structure	Small number of unified concepts ("everything is an object") at odds with outside world. Everything is composed from small number of primitives, but limits convenience. Structural commonality.	Files provide "large" common concepts, but details are open. Scripting based on small composable tools. Standard libraries and tools offer convenience.	Limited number of domain-specific concepts (sheet, formula, macro). Computation can be composed and formulas constructed using many convenient built-ins. Structural commonality.	Improvised mix of open "large" concepts (HTTP) and specific ones (DOM). Many convenient libraries and tools with low commonality and varying composability.	Notebook and cells as "large" concepts with code notions (Python) as "small" concepts. Composability primarily at code level, but not notebook level. Convenient libraries and tools.	Small number of unified concepts (functions, expressions) at odds with outside world. Composability at expression and type level. Limited set of convenience tools. Type classes for commonality.
→ Customizability	System can be customized at runtime. Much of the system is written in itself and can be modified from within itself. Extensibility achieved via object-oriented programming.	Explicit stage distinction between execution and building, but system is written using its own notation (C language) and can be modified and rebuilt from within itself. Limited modifiability at runtime.	Documents are editable during execution, but system itself cannot be modified. Adding only appends computations, but cannot modify existing ones.	Basic infrastructure (browser, protocols) are fixed. Individual applications can have a large degree of modifiability (via dynamic scripting). CSS provides powerful addressing.	System is fixed, but can theoretically be modified as open-source project with community. Programs cannot modify themselves, notebook or system at runtime.	Language is fixed, but can theoretically be modified as open-source project with community. Programs cannot modify themselves nor the system. Type classes allow extensibility at compile-time.
→ Complexity	Factoring using a rich class-based system covering system and application-level features. Basic automation (garbage collection) with more possible through libraries & via reflection.	Defines low-level infrastructure (hardware abstractions) and large object structure (files, processes); small-scale factoring and automation left to the user and/or application.	Fixed structure of formulas and grid. High-level language for formulas with automated re-computation. Programming-by-example provides next-step automation.	Factoring via high-level languages (JavaScript), rule-based systems (CSS) and standard interfaces (W3C specifications). Automation at basic level (garbage collection) and in declarative domains (CSS).	Complexity relegated to complex libraries (pandas, ML libraries, etc.) created outside the system. Basic language automation (GC) but no automatic recomputation in standard Jupyter setup.	Complexity factored using math-inspired type class hierarchies with type system support. Automates memory management (GC) and evaluation order (laziness).
→ Errors	Errors detected at runtime and can be corrected immediately in interactive editor/debugger. Further detection possible via engineering testing tools.	Error detection left to the system user. Low-level primitives make it possible to automate detection and response via custom mechanisms.	Slips caught at runtime, but no support for checking lapses or mistakes. Provides immediate feedback, making quick error correction possible.	Generally aims to do the best thing possible (automatic recovery) on errors. Direct error correction can be done in browser tools, but not permanent.	Slips caught at runtime. Limited checking of lapses or domain-specific mistakes. REPL-evaluation provides quick feedback, making quick error correction possible.	Strict error checking eliminates lapes and slips and some mistakes at compile time. Error correction done in text editor, based on non-trivial error messages.
→ Adoptability	Steep learning curve, but uniform design makes understanding reusable. End-users can progressively become programmers. Active community, but closed world and limited packages.	Requires background knowledge (system-level), but supported by active community. Openness allows integration with the external world; diversity of packages available.	Domain-focus on specific needs and graphical interface supports learning. End-users can progressively become programmers. No packaging mechanism, but wide range of samples and community available.	Web has a diversity of technologies; learnability is mainly achieved through community. The diversified web ecosystem allows for the integration with external systems.	Learnability is supported by focus on a specific domain, graphical interface and community. Notebooks can import a range of community packages and integrate with external systems.	Learning requires background knowledge (mathematics), but is supported by community and uniform design. Closed ecosystem, but with community and diversity of packages.

Charles University

Demo

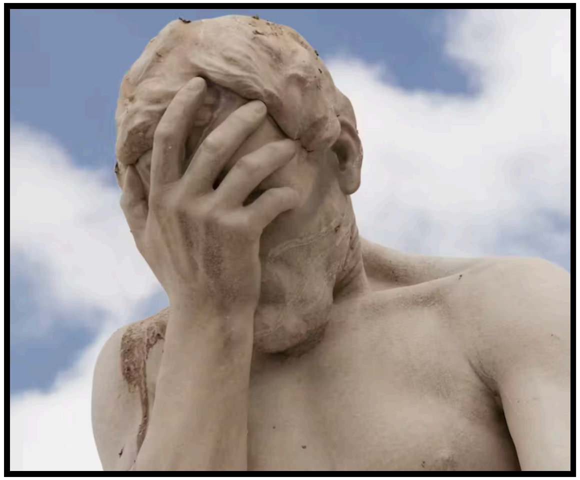
Excel data exploration basics

Charles University

The good and the bad

High usability

- Live exploratory programming
- Work with concrete values
- Learning from examples



High-profile errors

- "Growth in the time of debt" errors
- SEPT2, MARCH1 gene names
  (Septin, Membrane-Associated Ring Finger)

Charles University

Confusing terminology

Exploratory programming
Write, run, rethink with easy editing

Live programming
See results of your program immediately

Live coding
Run immediately, typically audio performance

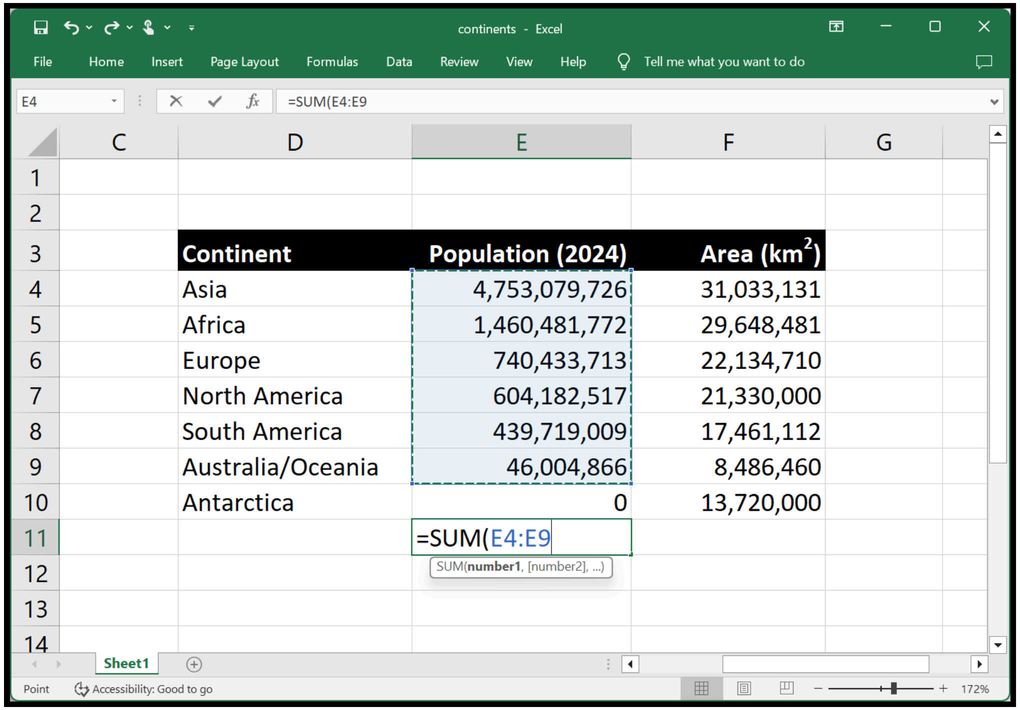
Interactive programming
Modify stateful programming system

Charles University

Spreadsheets are...

Exploratory - easy to fiddle with data

Live - you see results (almost) immediately



Charles University

Abstraction is hard

What is this ?
what are some
values

```
function population(pop, total) {
  return pop / total * 100;
}
```

did I get the
order wrong?

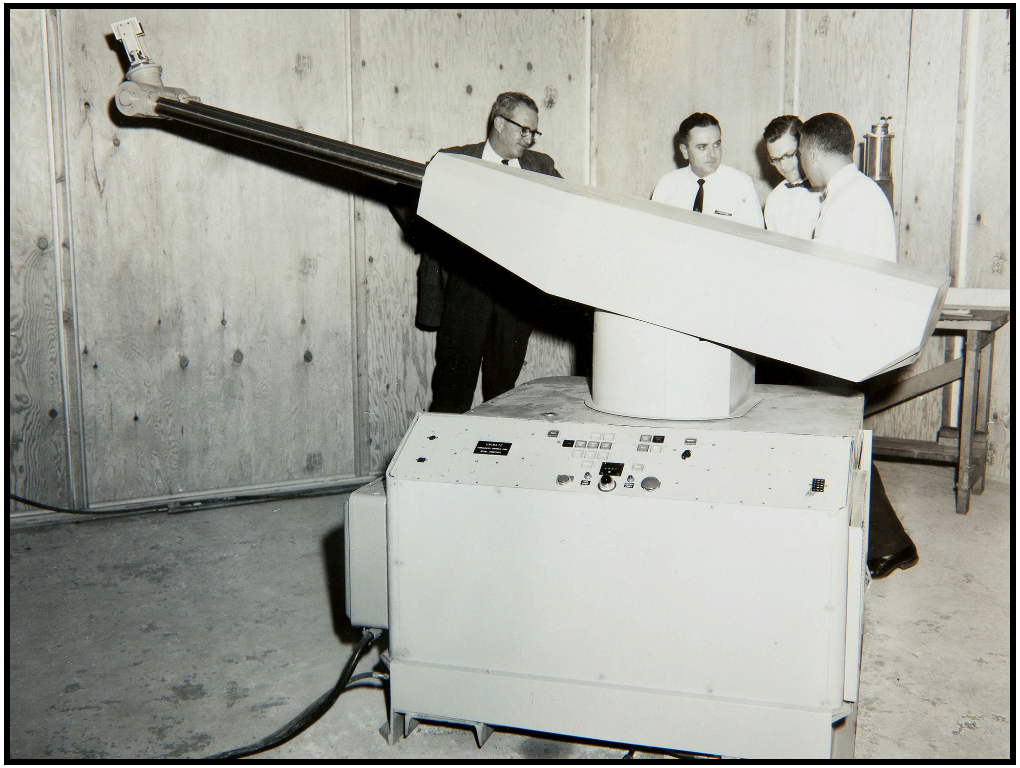
Drag-down for formulas makes abstraction easy

You only ever work with concrete values

Always see sample inputs & verify sample outputs

Charles University

Concreteness



Unimate industrial robot (1961)

Program by moving the robotic hand

Macro recording but done right

Charles University

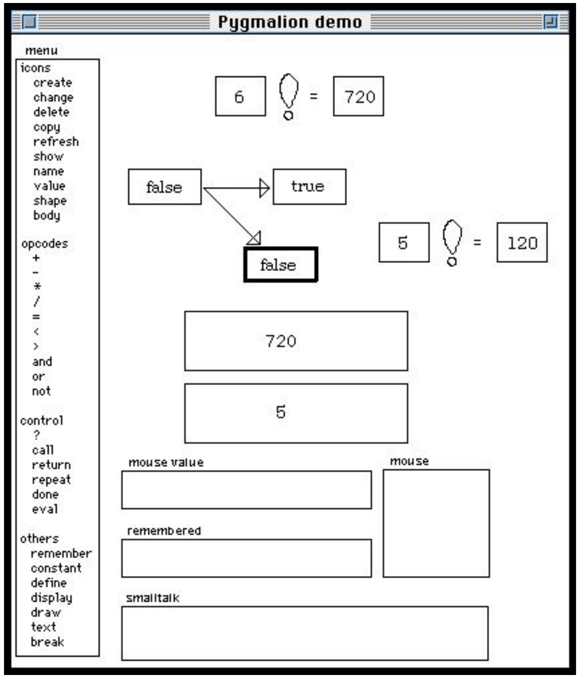
Concrete programming

Programming by demonstration

- Think macro recording
- How to generalize & re-apply
- "Drag down" in spreadsheets

Programming by example

- Generalize from input/output list
- Search for fitting program
- Also FlashFill in Excel



Charles University

Demo

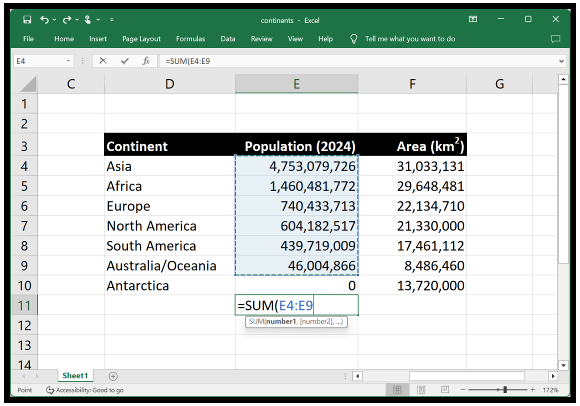
FlashFill in Excel

Charles University

How people learn Excel

From existing spreadsheets

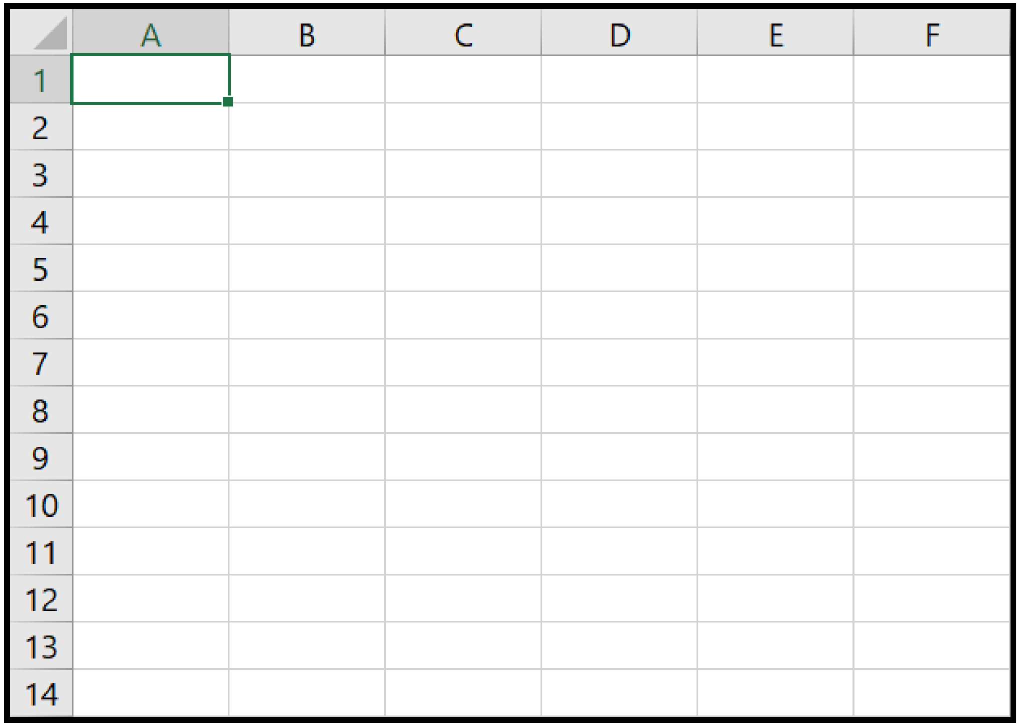
- View source of formulas
- Learn how functions work
- Logic needs to be visible!



Going to the expert

- Every office has Excel "guru"
- Needed for harder aspects
- Needed for use that does not have a "trace"

Charles University



The grid power!

Humans are good at working with space

Programs are not typically spatial...

Grid is limiting, but powerful concept

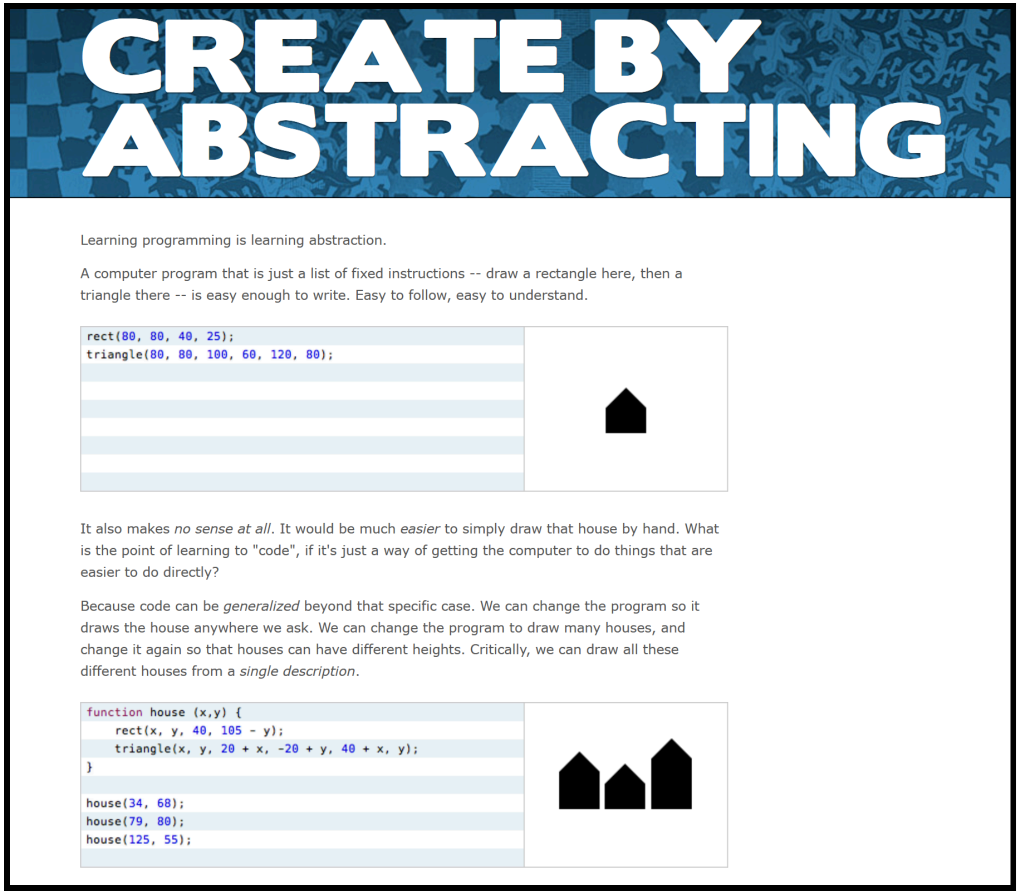
Charles University

TinyExcel

Learning from spreadsheets?

- More programming for non-programmers?
- Immediate live feedback is great!
- Abstractions from working with concrete values
- Programs should exist in understandable space

Charles University



CREATE BY ABSTRACTING

Learning programming is learning abstraction.

A computer program that is just a list of fixed instructions -- draw a rectangle here, then a triangle there -- is easy enough to write. Easy to follow, easy to understand.

```
rect(80, 80, 40, 25);
triangle(80, 80, 100, 60, 120, 80);
```

It also makes *no sense at all*. It would be much *easier* to simply draw that house by hand. What is the point of learning to "code", if it's just a way of getting the computer to do things that are easier to do directly?

Because code can be *generalized* beyond that specific case. We can change the program so it draws the house anywhere we ask. We can change the program to draw many houses, and change it again so that houses can have different heights. Critically, we can draw all these different houses from a *single description*.

```
function house (x,y) {
    rect(x, y, 40, 105 - y);
    triangle(x, y, 20 + x, -20 + y, 40 + x, y);
}

house(34, 68);
house(79, 80);
house(125, 55);
```

Could "normal" programming be more like this?

Demos by Bret Victor

Learnable Programming: Designing a programming system for understanding programs (online)

Charles University

TinyExcel

Scope of the tiny version

- Two-dimensional space with references
- "Drag-down" to apply formula to a column
- Relative and absolute cell references
- Incremental computational engine