NPRG077

TinyExcel: Tiny incremental spreadsheet system

Tomáš Petříček, 309 (3rd floor)

✉ petricek@d3s.mff.cuni.cz

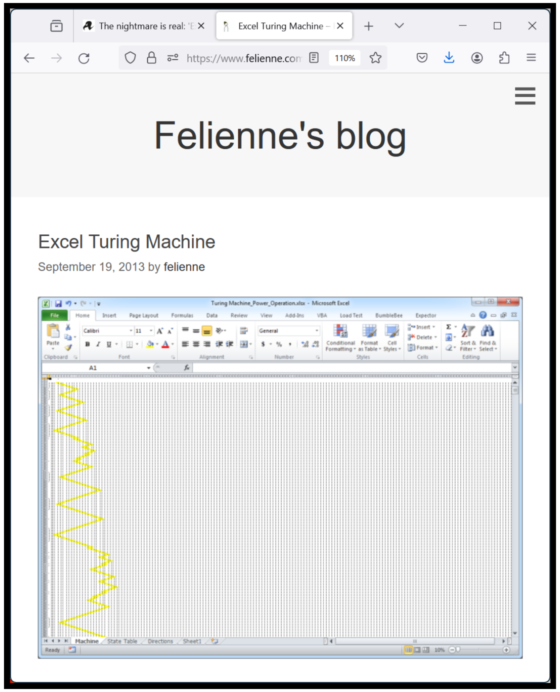
➔ https://tomasp.net | @tomaspetricek

➔ https://d3s.mff.cuni.cz/teaching/nprg077

Is Excel real programming?

It is Turing-complete!

Encoded using "drag-down"



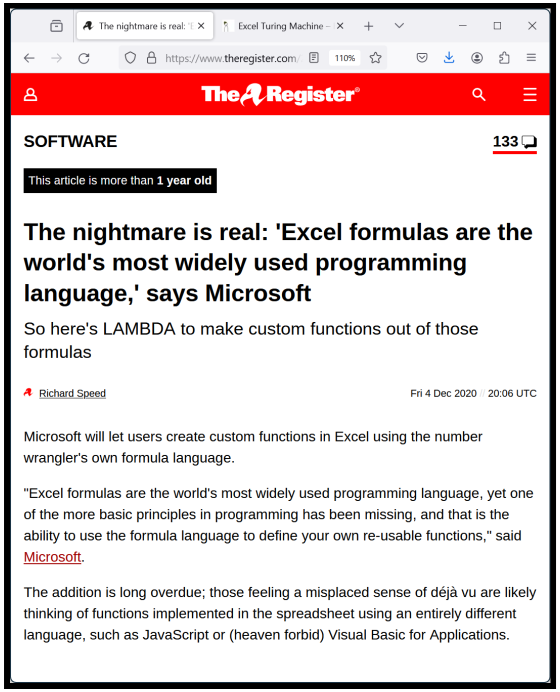
Felienne's blog

Excel Turing Machine

September 19, 2013 by felienne

It is widely-used!

Simple, but can do a lot...



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.

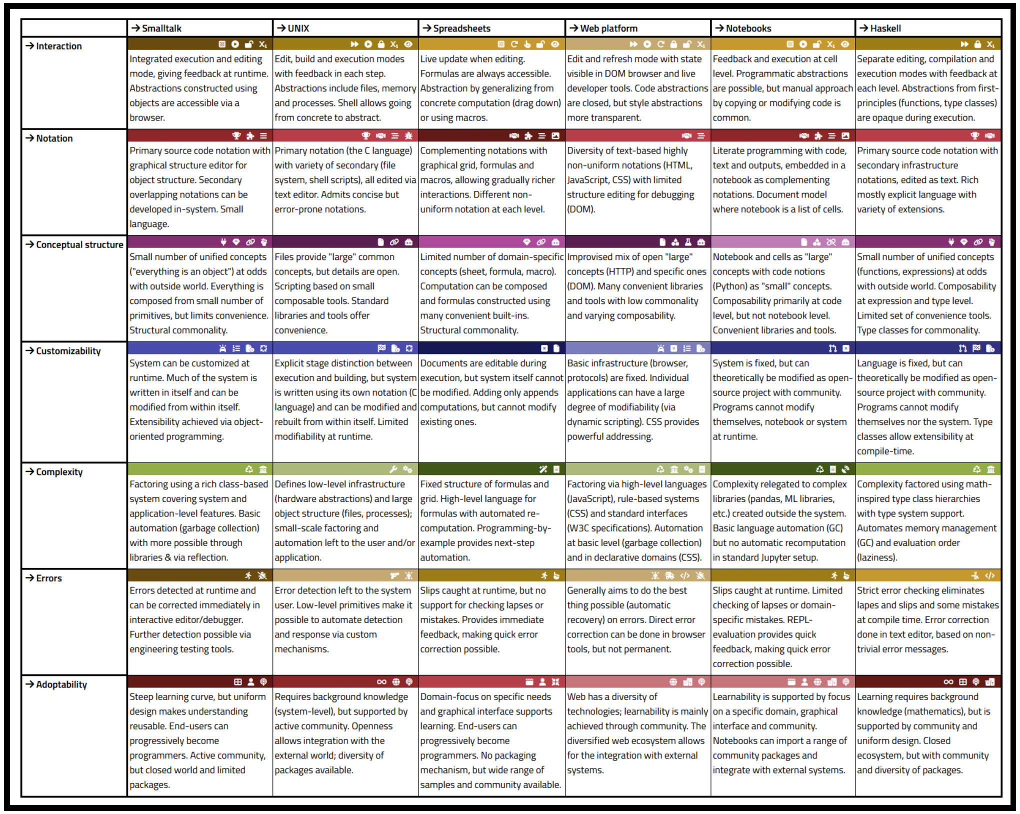
Charles University

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

Charles University



	→ 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.

Technical Dimensions of Programming Systems

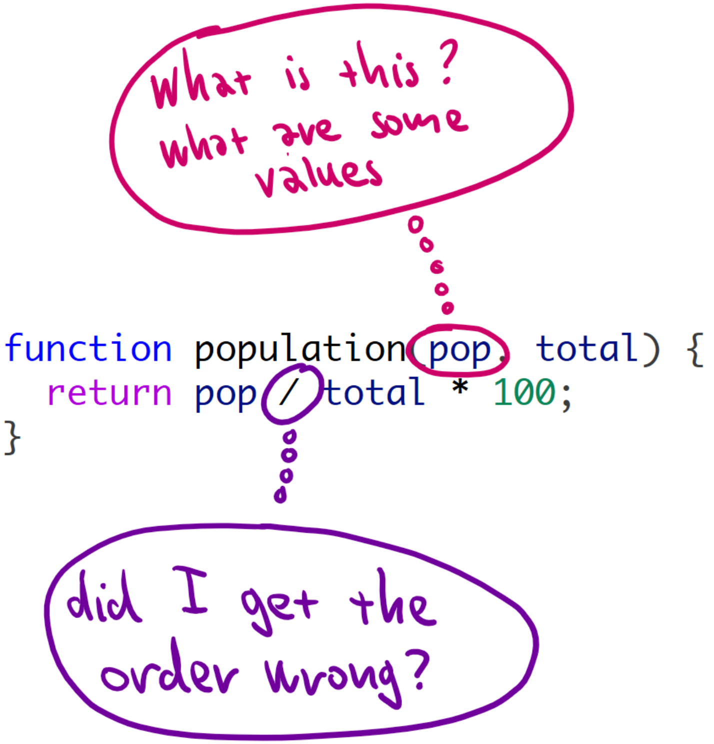
(Jakubovic et al., 2023)

What matters about stateful interactive systems?

Demo

Excel data exploration basics

Charles University



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

Abstraction is hard

Drag-down for formulas makes abstraction easy

You only ever work with concrete values

Always see sample inputs & verify sample outputs

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

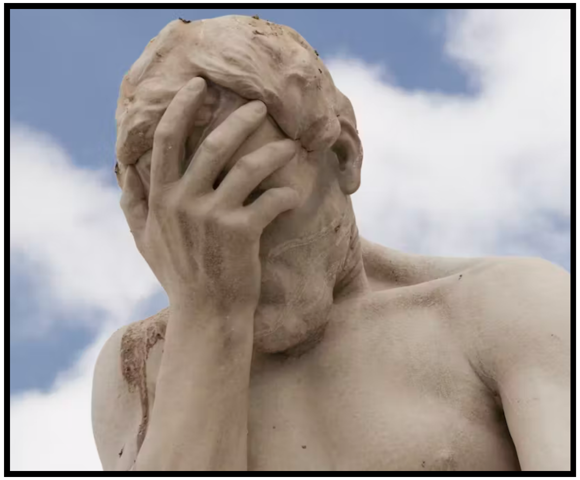
TinyExcel

Technical dimensions

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

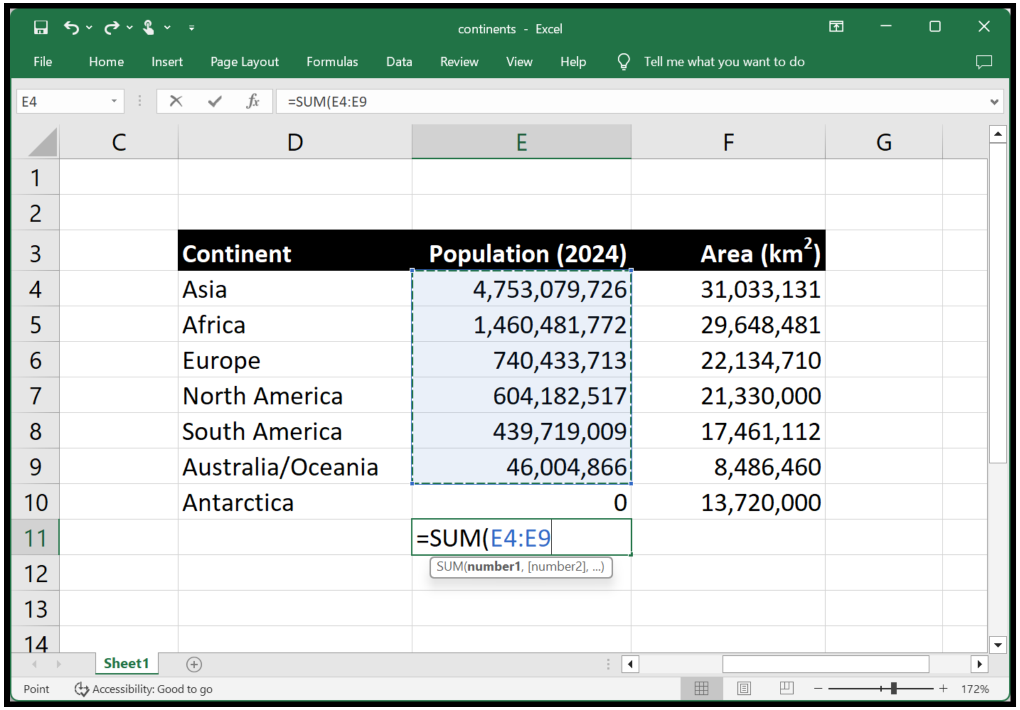
Confusing terminology

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



E4 =SUM(E4:E9

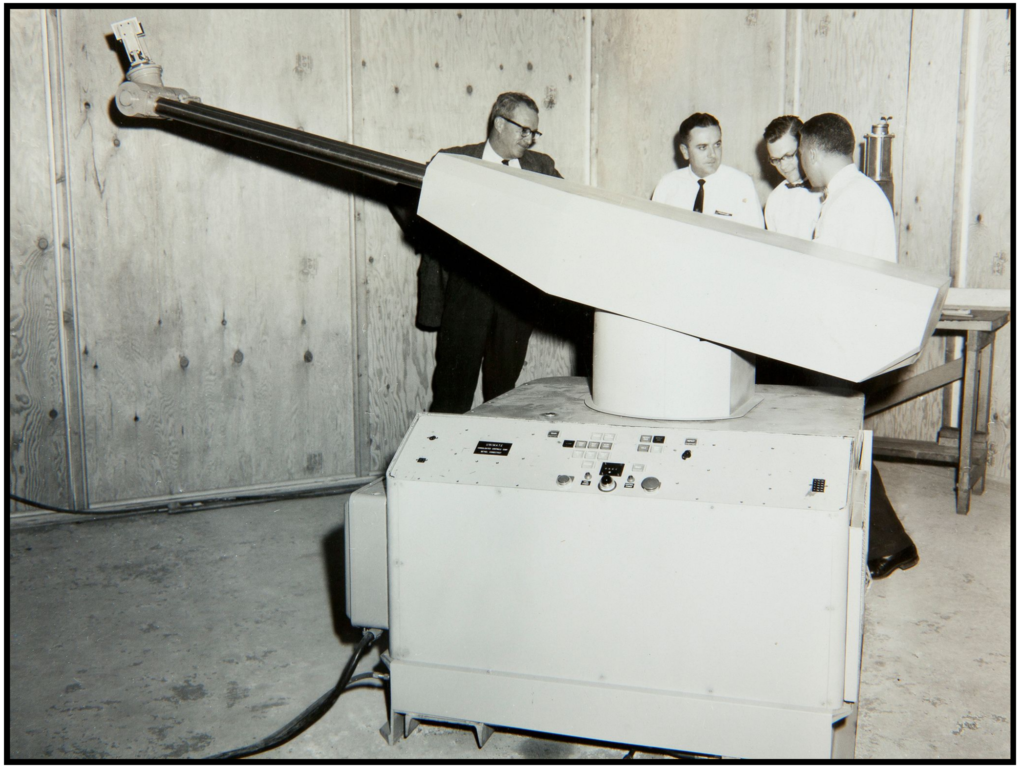
	C	D	E	F	G
1					
2					
3		Continent	Population (2024)	Area (km^2)	
4		Asia	4,753,079,726	31,033,131	
5		Africa	1,460,481,772	29,648,481	
6		Europe	740,433,713	22,134,710	
7		North America	604,182,517	21,330,000	
8		South America	439,719,009	17,461,112	
9		Australia/Oceania	46,004,866	8,486,460	
10		Antarctica	0	13,720,000	
11			=SUM(E4:E9		
12					
13					
14					

SUM(**number1**, [number2], ...)

Spreadsheets are…

Exploratory - easy to fiddle with data

Live - you see results (almost) immediately



Concreteness

Unimate industrial robot (1961)

Program by moving the robotic hand

Macro recording but done right

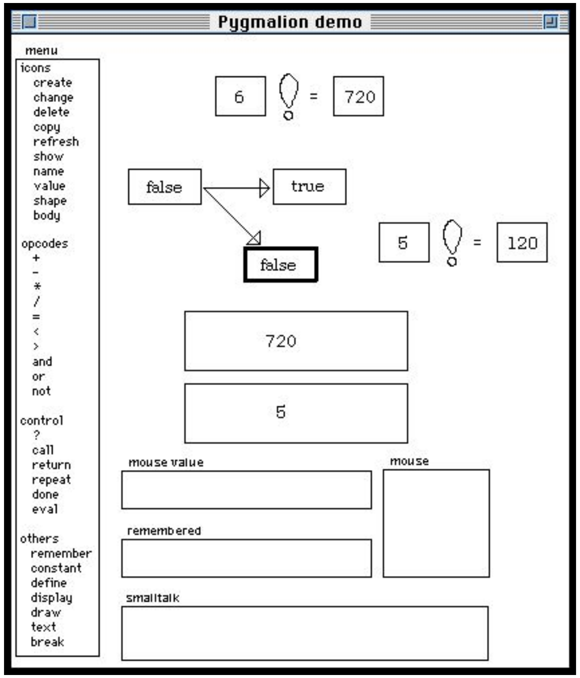
Concrete programming

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



Demo

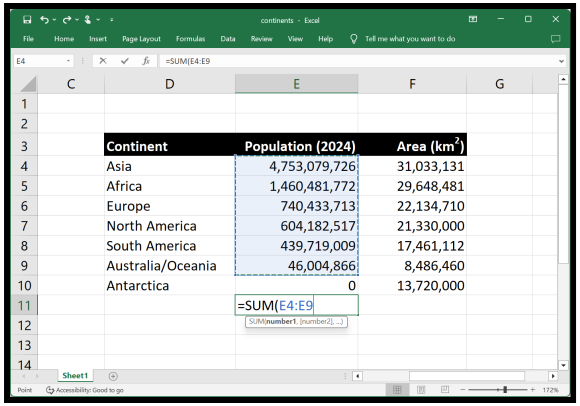
FlashFill in Excel

Charles University

How people learn Excel

From existing spreadsheets

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



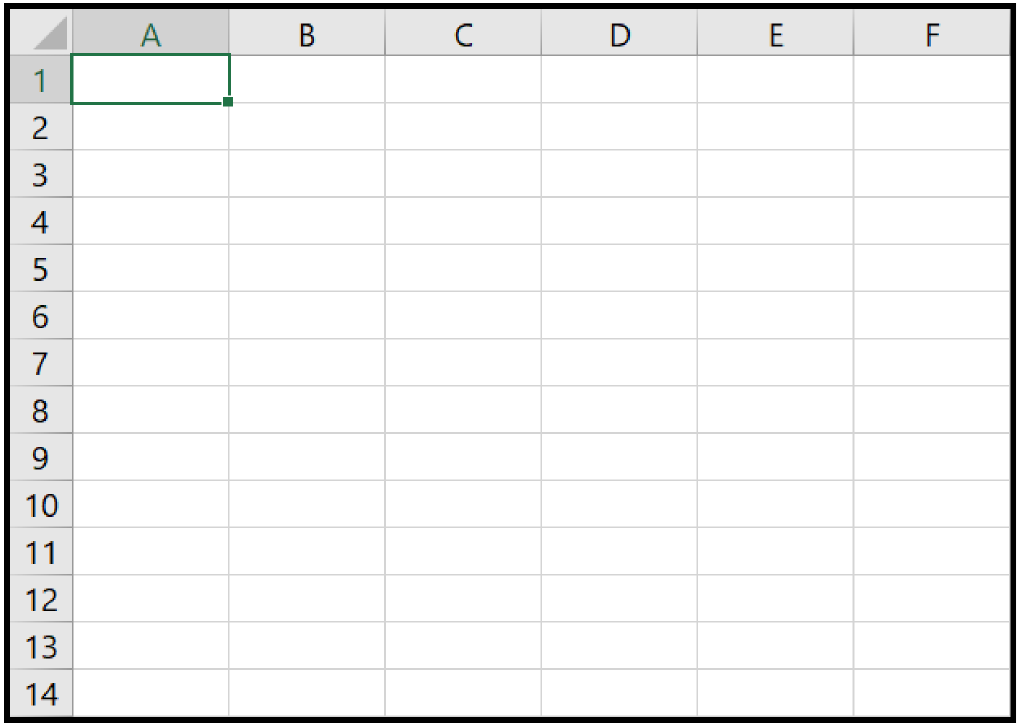
=SUM(E4:E9

	C	D	E	F	G
1					
2					
3		Continent	Population (2024)	Area (km^2)	
4		Asia	4,753,079,726	31,033,131	
5		Africa	1,460,481,772	29,648,481	
6		Europe	740,433,713	22,134,710	
7		North America	604,182,517	21,330,000	
8		South America	439,719,009	17,461,112	
9		Australia/Oceania	46,004,866	8,486,460	
10		Antarctica	0	13,720,000	
11			=SUM(E4:E9		
12					
13					
14					

SUM(**number1**, [number2], ...)

Going to the expert

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



	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

The grid power!

Humans are good at working with space

Programs are not typically spatial...

Grid is limiting, but powerful concept

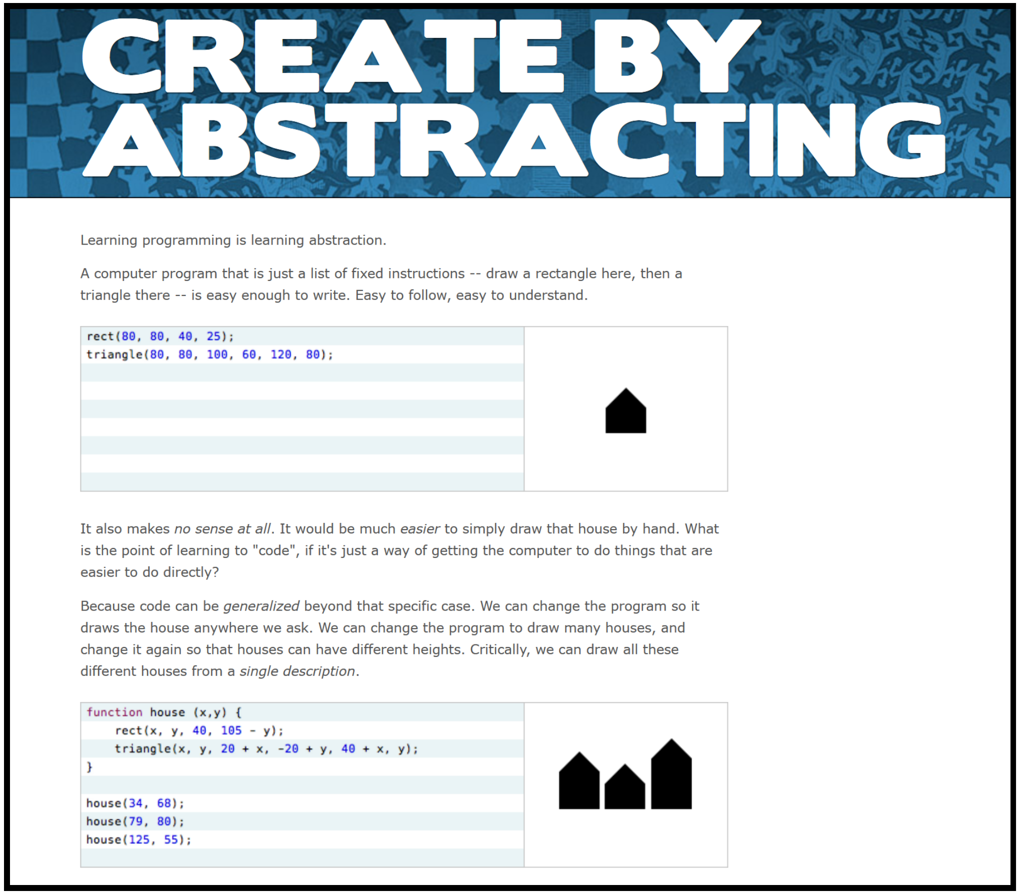
TinyExcel

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

Could "normal" programming be more like this?



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);
```

Demos by Bret Victor

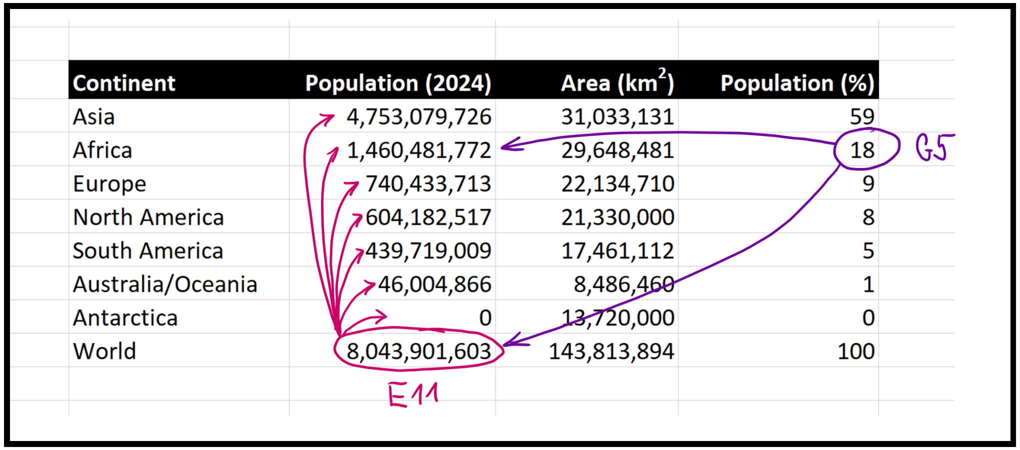
Learnable Programming: Designing a programming system for understanding programs (online)

Charles University

TinyExcel

Implementation techniques

Charles University



Continent	Population (2024)	Area (km^2)	Population (%)
Asia	4,753,079,726	31,033,131	59
Africa	1,460,481,772	29,648,481	18
Europe	740,433,713	22,134,710	9
North America	604,182,517	21,330,000	8
South America	439,719,009	17,461,112	5
Australia/Oceania	46,004,866	8,486,460	1
Antarctica	0	13,720,000	0
World	8,043,901,603	143,813,894	100

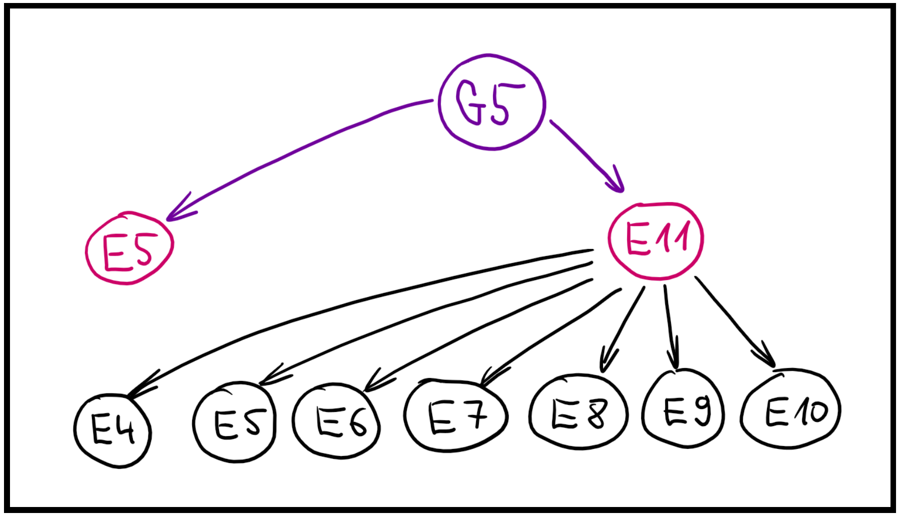
Inter-cell dependencies

In what order to evaluate sheet?

Avoid evaluating a cell repeatedly!

What to re-evaluate when cells change?

Charles University



Dependency graphs

Dependencies via cell and range references

Cyclic dependencies

Excel does a fixed maximal number of iterations

Explicit or implicit in code

Graph data structure vs. event listeners

Charles University

Reactive programming

Different implementations

- Functional Reactive Programming
- ReactiveX (rxjs, RxJava, Rx.Net)
- Elm software architecture

Implementation techniques

- Push-based - Changes propagated from source
- Pull-based - Update required by the consumer
- Builder-based - Computation to be instantiated

TinyExcel

Implementation techniques

- Naive non-cached recursive starting point
- Cell is as graph node with "Updated" event
- Depending nodes listen, recompute & notify
- Tricky error and update handling...

The F# language

What we need for Excel

Charles University

What we need to write Excel

Event handling

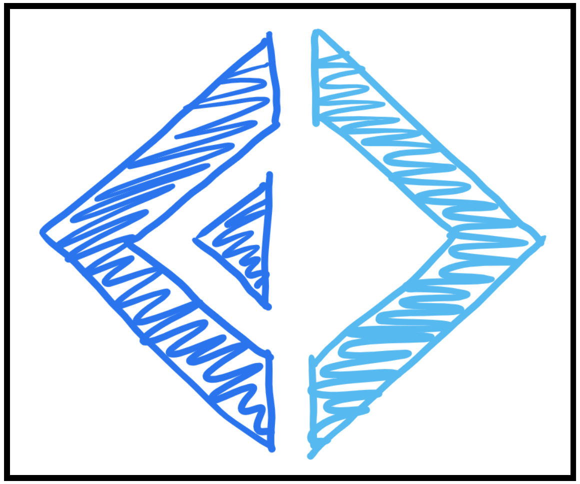
- F# events are objects (values)
- Can trigger & register handlers

More tips & tricks

- Collection processing
- Fancy patterns and active patterns

Finally a user interface?

- Would be nice, but setup costs high...
- Write sheet as HTML document & open



Charles University

Generating lists

List comprehensions with the yield keyword

```
let worldInfo =
  [ yield addr "A1", Const(String "Continent")
    yield addr "B1", Const(String "Population (thousands)")
    for i, (cont, pop) in Seq.indexed continents do
      yield addr ("A"+string(i+2)), Const(String cont)
      yield addr ("B"+string(i+2)), Const(Number pop) ]
```

- `yield` adds another item to the list
- `for` and other constructs to write generators
- `Seq.indexed` trick to get item index

Demo

Extending the List module

Charles University

F# Events

```
// Decares event value
let evt = Event<int>()

// Trigger event
evt.Trigger(1)
evt.Trigger(2)
evt.Trigger(3)

// Object for listening
evt.Publish

// Listen and print
evt.Publish.Add(fun n ->
  printfn "Got: %d" n)
```

Regular F# objects
Not special constructs

Corresond to `IObservable` in C#

Add and remove handlers using `AddHandler` and `RemoveHandler`

Demo

Working with F# events

Writing and opening HTML files

If you know C#, you can use other options too!

```
let demo () =
  let f = Path.GetTempFileName() + ".html"
  use wr = new StreamWriter(File.OpenWrite(f))
  wr.Write("""<html><body><h1>Hello world!</h1></body></html>""")
  wr.Close()
  Process.Start(f)
```

- `GetTempFileName` gives you a file in TEMP folder
- `use` to make sure stream gets closed on error
- `Process.Start` can (sometimes) open files too

TinyExcel

Implementation structure

Charles University

Simple start

```
// In column, row format
// e.g. A1 becomes (1, 1)
type Address = int * int

// Note error is a value!
type Value =
  | Number of int
  | String of string
  | Error of string

// Operators are functions
type Expr =
  | Const of Value
  | Reference of Address
  | Function of string * Expr list

// Using immutable F# map
type Sheet = Map<Address, Expr>
```

Standard ML-like expression language

References (instead of variables) are evaluated recursively

Sheet maps (filled) addresses to expressions

Version with the dependency graph

```
// Expression and value are
// mutable. Updated triggered
// when they change.
type CellNode =
  { mutable Value : Value
    mutable Expr : Expr
    Updated : Event<unit> }

// Immutable map
// of mutable cells
type LiveSheet =
  Map<Address, CellNode>
```

Value evaluated
on creation which
prevents circular refs

Expression stored
"drag down" expansion

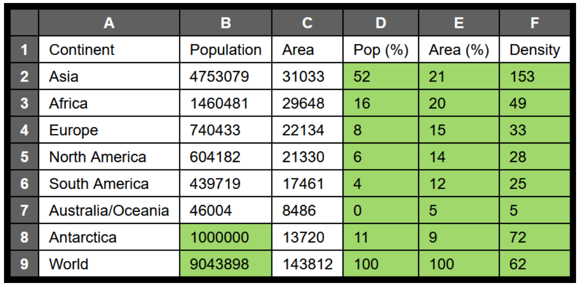
Updated event
to notify of changes

Advanced extensions

Ranges and array values

```
type Value = // (...)
  | Array of Value list

type Expr = // (...)
  | Range of Address * Address
```



	A	B	C	D	E	F
1	Continent	Population	Area	Pop (%)	Area (%)	Density
2	Asia	4753079	31033	52	21	153
3	Africa	1460481	29648	16	20	49
4	Europe	740433	22134	8	15	33
5	North America	604182	21330	6	14	28
6	South America	439719	17461	4	12	25
7	Australia/Oceania	46004	8486	0	5	5
8	Antarctica	1000000	13720	11	9	72
9	World	9043898	143812	100	100	62

Absolute addresses

```
type Index = Fixed of int | Normal of int
type RawAddress = int * int
type Address = Index * Index
```

Lab overview

TinyExcel step-by-step

Charles University

TinyExcel - Basic tasks

1. Simple expression evaluator
   With grid references by cell address
2. "Drag down" formula expanding
   Relocating relative references in formula
3. Reactive event-based structure
   Refactoring code to use graph nodes
4. Reactive event-based computation
   Adding update event handling
5. Rendering sheets as HTML pages
   First step towards a user interface

TinyExcel - Bonus and super tasks

1. Absolute and relative addresses
   Alongside with improved "drag down"
2. Adding range selection and array values
   Required for the SUM function
3. Adding change visualization
   Tracking and showing what has changed
4. Full support for live editing
   Updating dependencies in the dependency graph

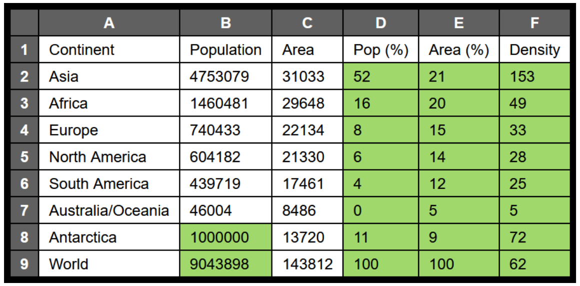
Closing

Tiny incremental spreadsheet system

Where can you use this...

Financial systems

- Live financial models
- Incremental computation with dependency graph



	A	B	C	D	E	F
1	Continent	Population	Area	Pop (%)	Area (%)	Density
2	Asia	4753079	31033	52	21	153
3	Africa	1460481	29648	16	20	49
4	Europe	740433	22134	8	15	33
5	North America	604182	21330	6	14	28
6	South America	439719	17461	4	12	25
7	Australia/Oceania	46004	8486	0	5	5
8	Antarctica	1000000	13720	11	9	72
9	World	9043898	143812	100	100	62

Interesting programming systems

- Live programming systems
- Future more usable programming tools!

Charles University

Conclusions

A tiny incremental spreadsheet system

- Computation as dependency graph
- Working with two-dimensional grid
- Good old (ML-like) expressions

Tomáš Petříček, 309 (3rd floor)
✉ petricek@d3s.mff.cuni.cz
➜ https://tomasp.net | @tomaspetricek
➜ https://d3s.mff.cuni.cz/teaching/nprg077

Charles University

https://direct.mit.edu/books/book/3071/Spreadsheet-Implementation-TechnologyBasics-and (hard to get...)

https://www.theregister.com/2020/12/04/microsoft_excel_lambda

https://www.felienne.com/archives/2974

https://arxiv.org/ftp/arxiv/papers/1807/1807.08578.pdf

https://theconversation.com/the-reinhart-rogoff-error-or-how-not-to-excel-at-economics-13646

https://genomebiology.biomedcentral.com/articles/10.1186/s1305
016-1044-7

https://advait.org/publications-web/sarkar-2018-spreadsheet-learning