

Functional programming in F#

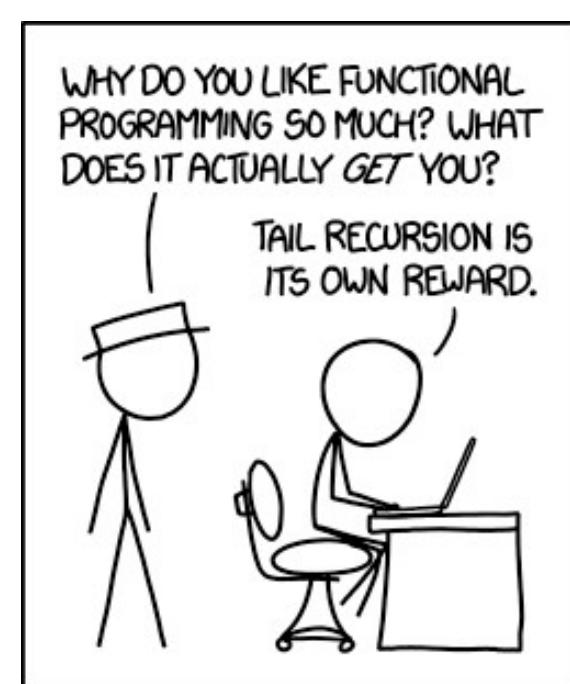
Jonas Juselius

11.10.2018



Functional programming

- History of programming
- Programming languages
- Complexity
- Lambda calculus
- Why F# and .NET Core?



A Brief, Incomplete, and Mostly Wrong History of Programming Languages

1801 - Joseph Marie Jacquard uses punch cards to instruct a loom to weave "hello, world" into a tapestry.

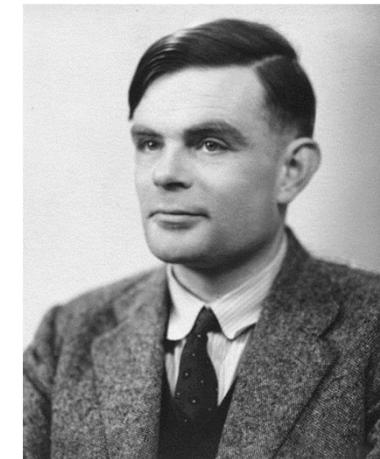
1842 - Ada Lovelace writes the first program. She is hampered in her efforts by the minor inconvenience that she doesn't have any actual computers to run her code.



<http://james-iry.blogspot.com/2009/05/brief-incomplete-and-mostly-wrong.html>

The fundamentalists

1936 - Alan Turing invents every programming language that will ever be but is shanghaied by British Intelligence to be 007 before he can patent them.

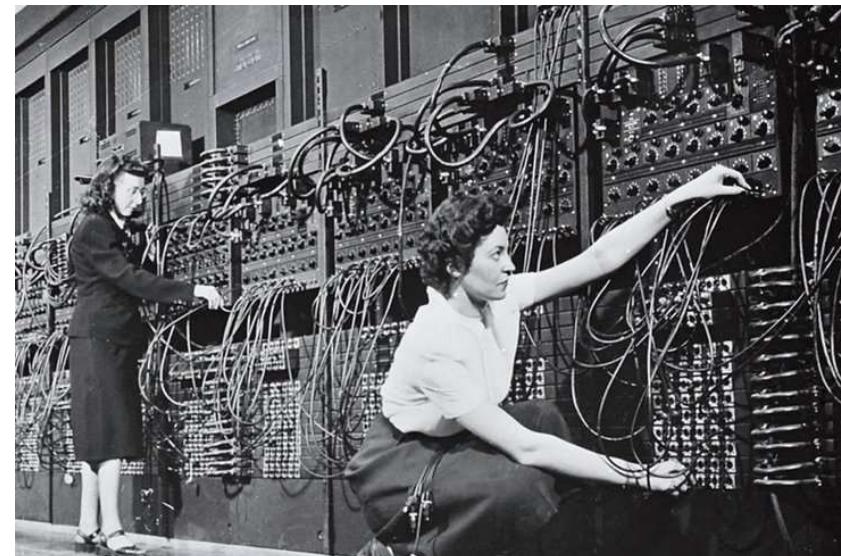
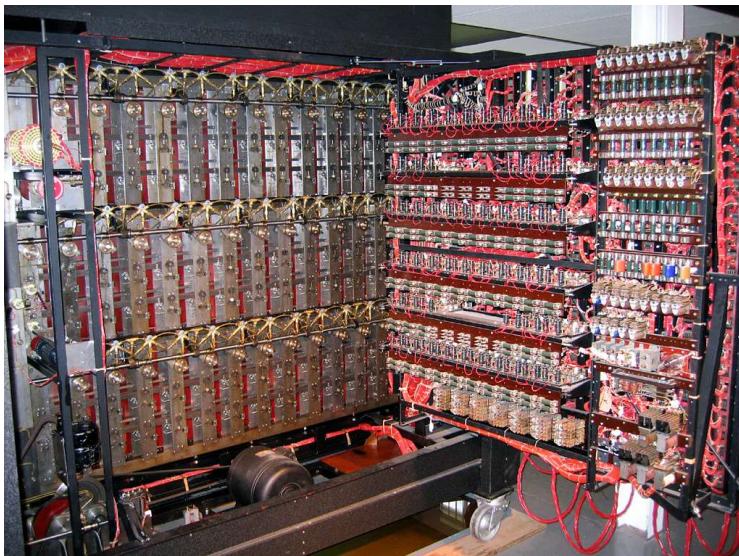


1936 - Alonzo Church also invents every language that will ever be but does it better. His lambda calculus is ignored because it is insufficiently C-like. Lambdas are relegated to relative obscurity until Java makes them popular by not having them.



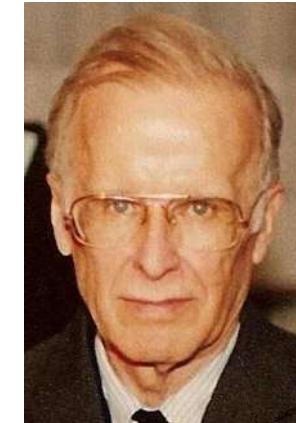
Wire, wire on the wall

1940s - Various "computers" are "programmed" using direct wiring and switches. Engineers do this in order to avoid the tabs vs spaces debate.



Vi vs. Emacs

1957 - John Backus and IBM create FORTRAN. There's nothing funny about IBM or FORTRAN.



1958 - John McCarthy and Paul Graham invent LISP. Due to high costs caused by a post-war depletion of the strategic parentheses reserve LISP never becomes popular.



Greenspun's tenth rule of programming:

Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of Common Lisp.

Generation 64

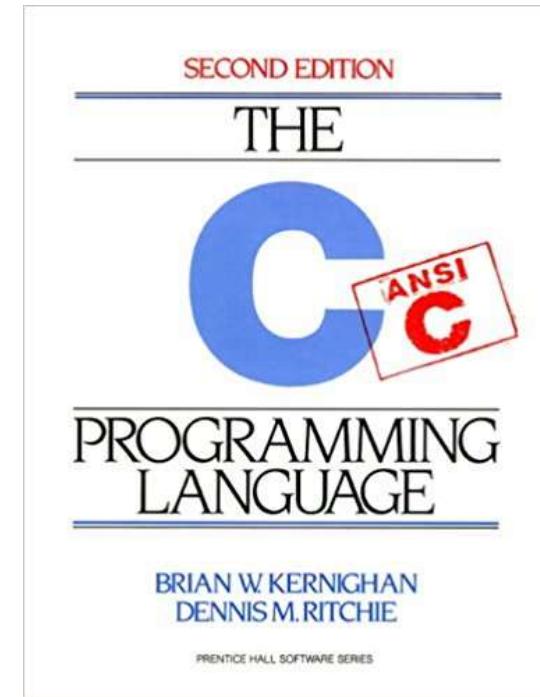
1964 - John Kemeny and Thomas Kurtz create BASIC, an unstructured programming language for non-computer scientists.

1965 - Kemeny and Kurtz go to 1964.



C

1972 - Dennis Ritchie invents a powerful gun that shoots both forward and backward simultaneously. Not satisfied with the number of deaths and permanent maimings from that invention he invents C and Unix.



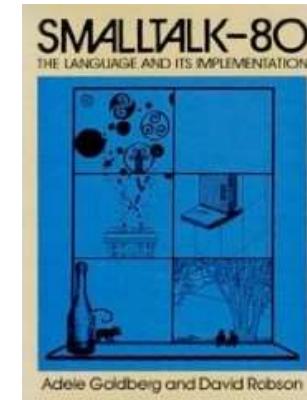
ML -> F#

1973 - Robin Milner creates ML, a language based on the M&M type theory. ML begets SML which has a formally specified semantics. When asked for a formal semantics of the formal semantics Milner's head explodes. Other well known languages in the ML family include OCaml, F# and Haskell.

$$\begin{array}{ll} (var) & \frac{(x:\sigma) \in A}{A \vdash x : \sigma} \\ (\rightarrow E) & \frac{A \vdash E : \tau' \rightarrow \tau \quad A \vdash F : \tau'}{A \vdash EF : \tau} \\ (\rightarrow I) & \frac{A_x, x:\tau' \vdash E : \tau}{A \vdash \lambda x.E : \tau' \rightarrow \tau} \\ (let) & \frac{A \vdash E : \sigma \quad A_x, x:\sigma \vdash F : \tau}{A \vdash (\text{let } x = E \text{ in } F) : \tau} \\ (\forall E) & \frac{A \vdash E : \forall \alpha^\kappa. \sigma \quad C \in C^\kappa}{A \vdash E : [C/\alpha^\kappa]\sigma} \\ (\forall I) & \frac{A \vdash E : \sigma \quad \alpha^\kappa \notin CV(A)}{A \vdash E : \forall \alpha^\kappa. \sigma} \end{array}$$


SmallTalk

1980 - Alan Kay creates Smalltalk and invents the term "object oriented." When asked what that means he replies, "Smalltalk programs are just objects." When asked what objects are made of he replies, "objects." When asked again he says "look, it's all objects all the way down. Until you reach turtles."

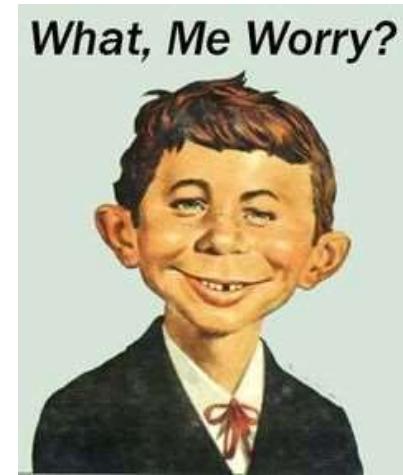


Frequently Questioned Answers

1983 - Bjarne Stroustrup bolts everything he's ever heard of onto C to create C++.

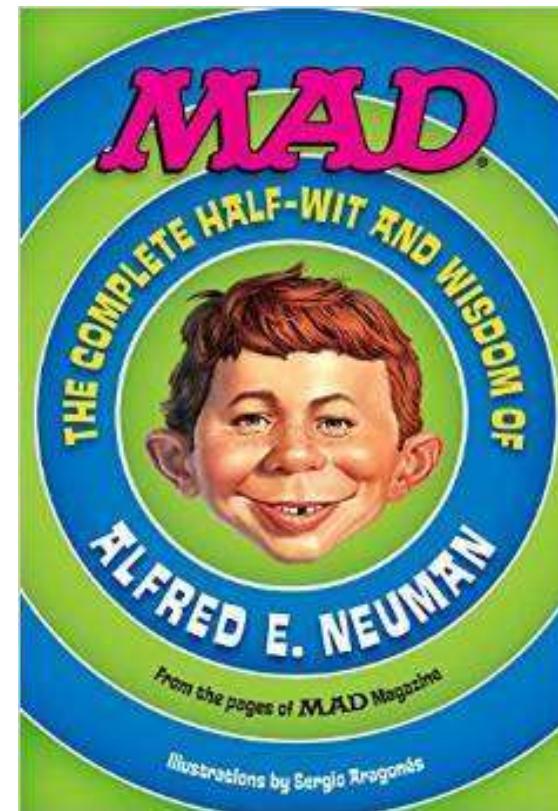
1996 - James Gosling invents Java. Java is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Sun loudly heralds Java's novelty.

2001 - Anders Hejlsberg invents C#. C# is a relatively verbose, garbage collected, class based, statically typed, single dispatch, object oriented language with single implementation inheritance and multiple interface inheritance. Microsoft loudly heralds C#'s novelty.



JavaScript

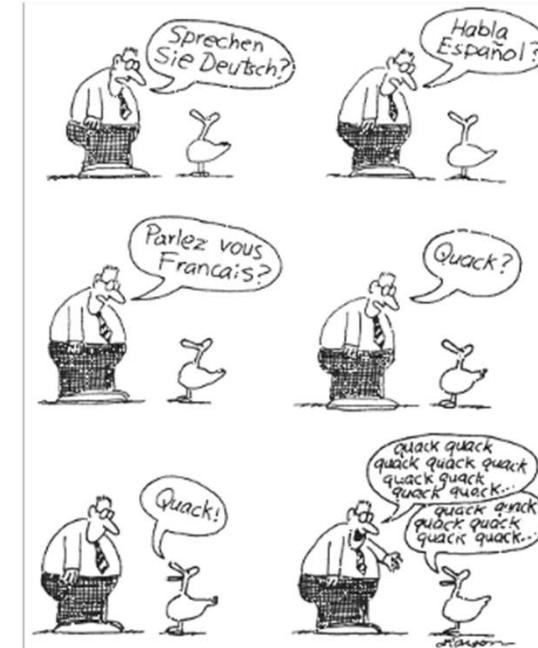
1995 - Brendan Eich reads up on every mistake ever made in designing a programming language, invents a few more, and creates JavaScript.



Programming languages are *tools* for *aiding* our thought, and expressing our intent to computers.

*Fader Dator, som är i skyn.
Helgad vare min skärm.
Tillkomme mitt tangentbord.
Ske min vilja, såsom i editorn
Så ock i output.
Vår dagliga sprint giv oss i dag,
Och förlåt oss våra misstag,
Trots att vi icke förlåta dem som programmerat fel.
Låtom oss icke ha långa svarstider
Och fräls oss från Virus.
Ty Abstraktionen är min
Och Applikationen och Rekursionen
I Evighet.*

Enter



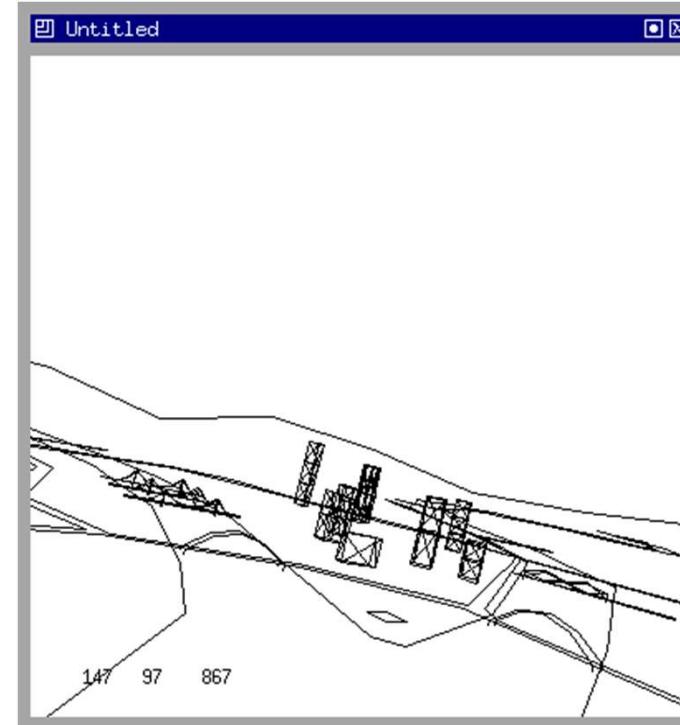
It's not about *what*, it's about *how*!

```
#include <math.h>
#include <sys/time.h>
#include <X11/Xlib.h>
#include <X11/keysym.h>
double L ,o ,P
,_=dt,T,Z,d=1,d,
z[999],E,h= 8,I,
J,K,w[999],M,m,O
,n[999],j=33e-3,i=
1E3,r,t,u,v ,W,S=
74.5,l=221,X=7.26,
a,B,A=32.2,c, F,H;
int N,q, C, y,p,U;
Window z; char f[52]
; GC k: main() { Display(e,
XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC ( e,z,0,0),BlackPixel(e,0))
; scanf("%lf%lf%lf",y +n,w+y, y+=1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400
,0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); T=sin(O){ struct timeval G{ 0,dt*1e6}
; K= cos(j); N=le4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*I; O+=D*_*F/ K+d/K*E* ; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D- *_F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D *; pcy; ) T=p[s]+i; E=c+p[w]; D=[p]-l; K=d*m-B*T-H*; if(p [n]+w[ p]+p[s
]= 0|K <fabs (W*T*r-I*E +D*B) |fabs(D+m *DZ *T-E )> K) N=le4; else q=W/K *4E2+e2; C= 2E2+e2/ K
*D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } l+=_* ( X*t +P*M+m*1); T=X*X+ 1*l+M *M;
XDrawString(e,z,k ,20,380,f,17); Dev/1*15; i+=(B *_M*r -X*z)_* ; for(; XPending(e); u *=CS!=N)
XEvent z; XNextEvent(e,&z);
+++((N==XLookupKeySym
(&z.xkey,0))-IT?
N-LT? UP-N?6 E:6
J:& u: sh: --*(

DN -N N-DT ?N=-
RT?su & W:sh:&J
); ) m=15*F/1;
c+=(I=M/ 1,1*H
+I*M+a*X)* ; H
=A*rr+v*X-F*1+(

E.=1-X*4.9/1,t
=T*m/32-IT/24
)/S; K=F*MM+
h* 1e4/1-(T+
E*5*T*E)/3e2
)/S-X*d-B*A;
a=2.63 /1*d;
X=( d*1-T/S
*(.19*E +a
*.64+0/1e3
)-M* v +A*
Z)*; l +=_
K *_; W=d;
sprintf(f,
"%5d %3d"
"%7d",p =1
/1.7, (C=9E3+
O*57.3)*0550,(int)i); di=T* (.45-14/1*
X-a*130-J* .14)* _/125e2+F* *v; P=(T*(47
*I-m* 52+E*94 *D-t*.38-u*.21*E) /1e2+W*
179*v)/2312; select(p=0,0,0,0,&G); v=(

W*E-T* (.63*m-I*.0864*m*E*19-D*25-.11*u
)/107e2)*_; D=cos(o); E=sin(o); ) }
```



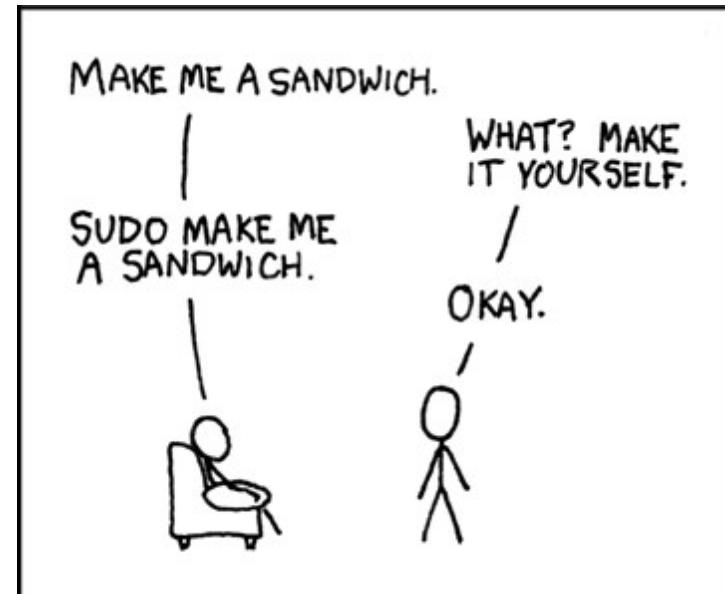
The dyadic paradigm

Declarative programming

- Focus on *what* rather than *how*
- Describes the problem to be solved
- Reflects how humans operate

Imperative programming

- Describes *how* a problem is solved
- Reflects how processors operate
- Use statements to change *state*
- Large degree of fine-grained control



Sudoku

```
sudoku(Rows) :-  
    length(Rows, 9),  
    maplist(same_length(Rows), Rows),  
    append(Rows, Vs), Vs ins 1..9,  
    maplist(all_distinct, Rows),  
    transpose(Rows, Columns),  
    maplist(all_distinct, Columns),  
    Rows = [As,Bs,Cs,Ds,Es,Fs,Gs,Hs,Is],  
    blocks(As, Bs, Cs),  
    blocks(Ds, Es, Fs),  
    blocks(Gs, Hs, Is).  
blocks([], [], []).  
blocks([N1,N2,N3|Ns1], [N4,N5,N6|Ns2], [N7,N8,N9|Ns3]) :-  
    all_distinct([N1,N2,N3,N4,N5,N6,N7,N8,N9]), blocks(Ns1, Ns2, Ns3).
```

Exercise: Implement this in C# and compare



SWI Prolog

Sequel

```
TRANSFORM Sum(a.DocTotal) AS Income  
SELECT b.project_title  
FROM (  
    SELECT a.projectid, a.DocNo, a.CustomerID, a.DocTotal, a.DocDate  
    FROM Invoice AS a  
    INNER JOIN InvLines AS b  
    ON a.DocKey = b.DocKey  
    WHERE Year(a.DocDate) = 1999  
) AS a  
INNER JOIN AdvProjects AS b  
ON a.projectid = b.projectid  
GROUP BY b.project_title  
PIVOT Month(a.DocDate);
```

Exercise: Implement this in Java and compare





The world's #1 programming language!

C11 <L> TOTAL			
	A	B	C
	ITEM	NO.	UNIT
1	MUCK RAKE	43	12.95
2	BUZZ CUT	15	6.75
3	TOE TONER	250	49.95
4	EYE SNUFF	2	4.95
5			-----
6			SUBTOTAL
7			13155.50
8			9.75% TAX
9			1282.66
10			-----
11			TOTAL
12			14438.16
13			
14			
15			
16			
17			
18			
19			
20			

C++, anyone?



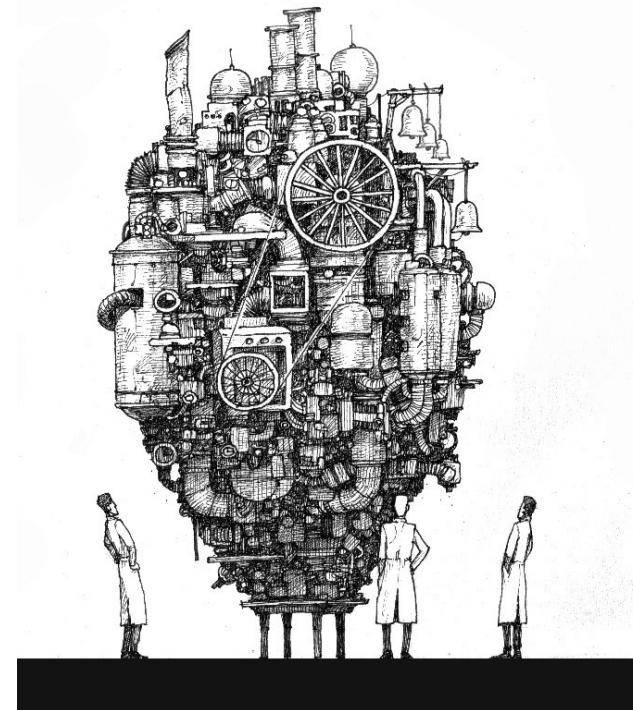
Out of the tar pit (Moseley & Marks)

*“There are two ways of constructing a software design:
One way is to make it so simple that there
are obviously no deficiencies, and the other way is to
make it so complicated that there are
no obvious deficiencies. The first method is far more
difficult”*

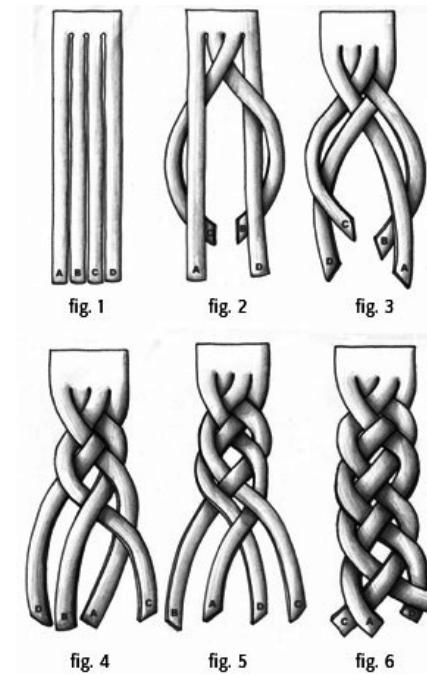
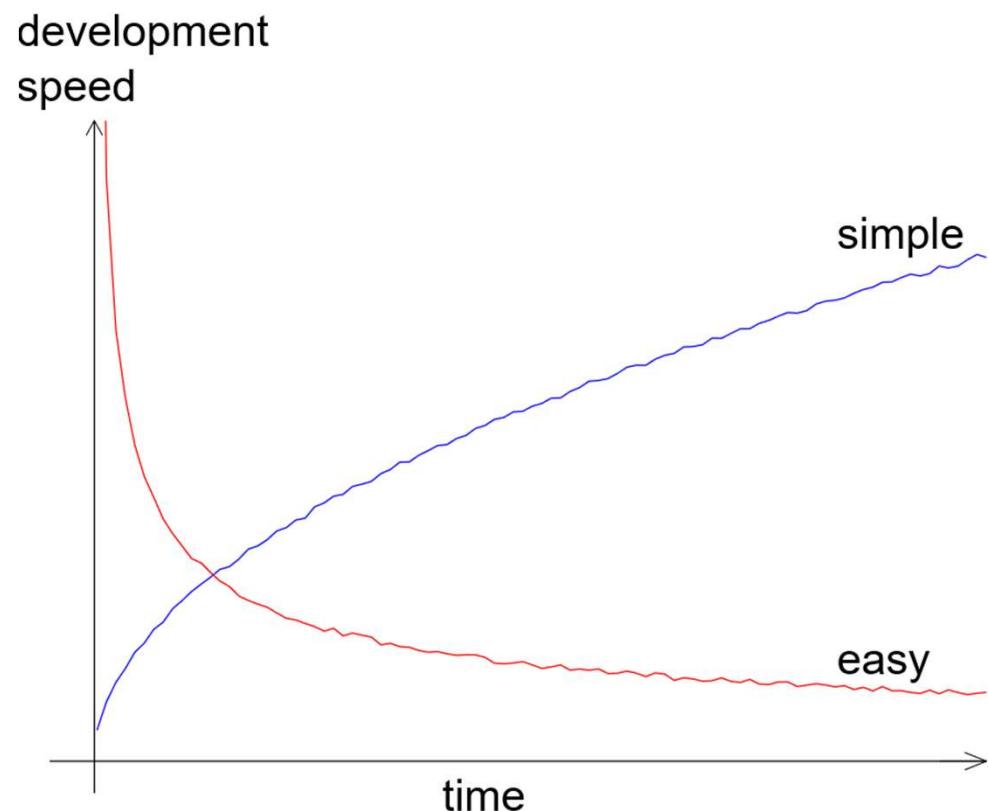
C.A.R. Hoare (Turing award lecture 1980)

Complexity

1. *Inherent* to the problem
 - Problem specification
 - Algorithms
2. *Incidental* to the solution
 - Programming languages
 - Tools, editors, debuggers
 - Hardware
 - Unintended interactions
 - I/O



Simple vs. easy



Really?

$x = x + 1$



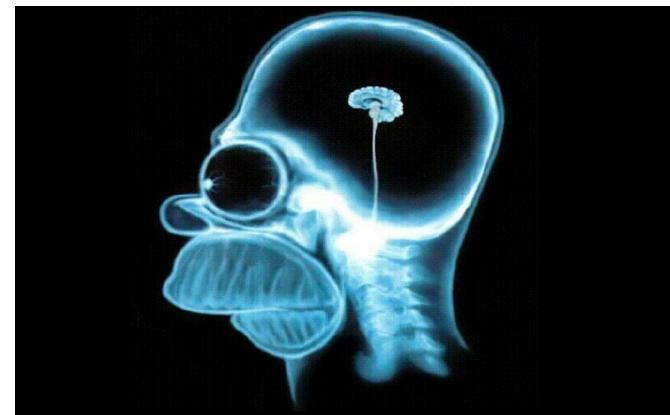
No.

$x \neq x + 1$



$$f(x) = (x + 1)(x + 2)$$

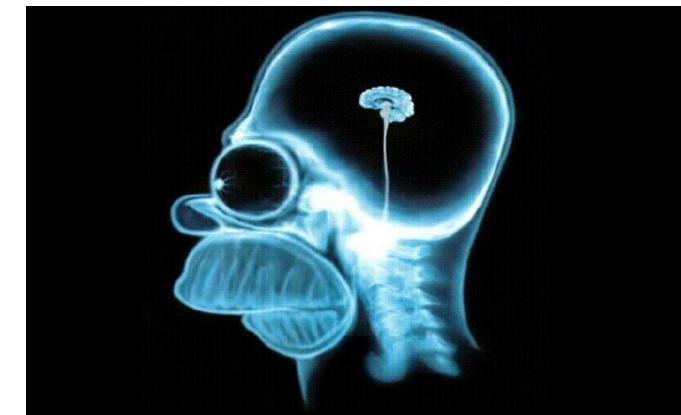
```
def f(x):  
    x = x + 1  
    y = x + 1  
    return x * y
```



$$f(x) = (x + 1)(x + 1)$$

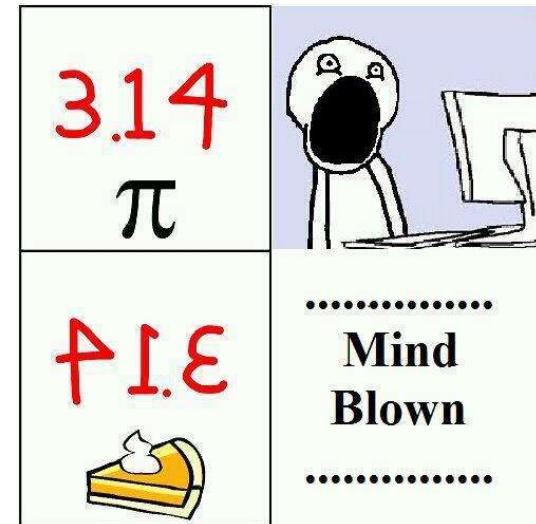
```
def f(x):  
    y = x + 1  
    x = x + 1  
    return x * y
```

Implicit time



$$f(x) = (x + 1)(x + 2)$$

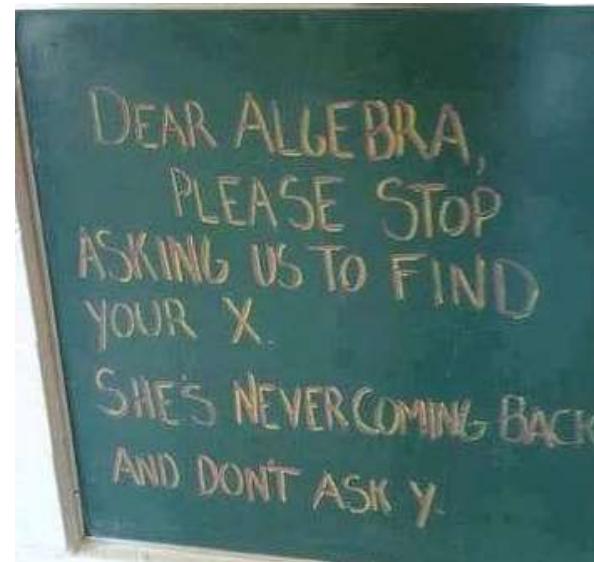
```
def f(x):  
    a = x + 1  
    b = a + 1  
    return a * b
```



Pure functions

- A function takes **one** input and returns **one** output
- Always.
- Functions don't have access to variables, state or I/O
- Functions are boring

Boring is good!



Procedures and methods



Variable updates are dangerous!

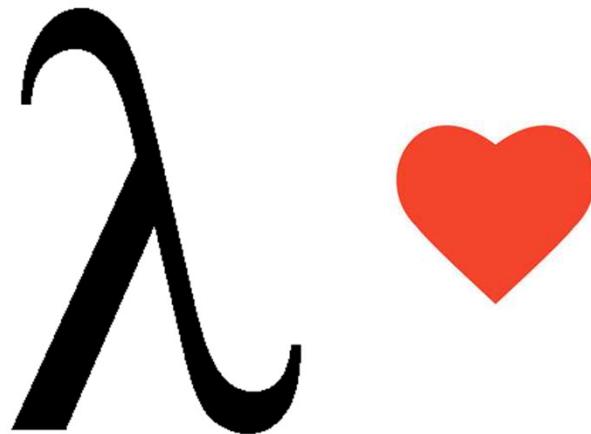


*Me at the office,
preparing to mutate a
variable*

Lambda calculus

λ -calculus is a formal system in mathematical logic for expressing computation based on function abstraction and application using variable binding and substitution.

λ -calculus It is a universal model of computation that can be used to simulate any Turing machine. λ -calculus is also Pac-Man complete.



```
fun n y -> n
```

Lambda calculus in practice

- No difference between functions and data
- Higher-order functions
- Pure functions without side-effects
- No variables

It's easier to understand and reason about pure code!

(But it takes practice!)



Komposisjon

Hvordan bygge kompleksitet fra enkle elementer?

Funksjoner:

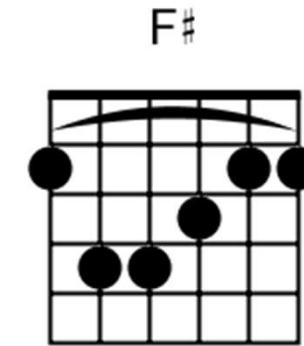
- $f(g(x)) \equiv f \circ g(x) = h(x)$
- `print(cos(exp(-1)))` $\equiv -1 |> \exp |> \cos |> \text{print}$

Data:

- `[1, 2, 3] + [4, 5]`
- `{'icecream': 'vanilla'} + {'vehicle': 'van'}`

Objekt:

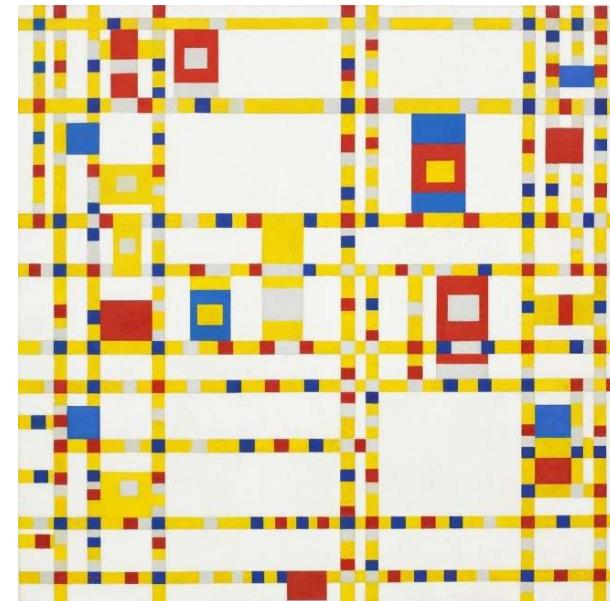
- `IceCream ('vanilla') + Van ()` = ???



Abstraksjon

Abstraksjon er et begrep for det som kun kan fattes gjennom tankene, uten direkte forbindelse med livet. Det benyttes om tenkemåter der en skiller ut enkelte elementer, for å betrakte resten. Det er en måte å komme fram til fellestrekks eller felles egenskaper og klassifiseringer, såkalte universalia, på.

- Funksjoner
- Typeteori
- Kategoriteori
- Algebraiske datatyper
- Curry-Howard isomorfismen



Broadway Boogie Woogie (Mondrian 1943)

OOP Pattern/Principle	FP Equivalent
Single responsibility principle	Functions
Open/Closed principle	Functions
Dependency inversion principle	Functions
Interface segregation principle	Functions
Factory pattern	Functions
Strategy pattern	Functions
Decorator pattern	Yeah, functions
Visitor pattern	You guessed it, functions!

“Design patterns are bug reports against your programming language”

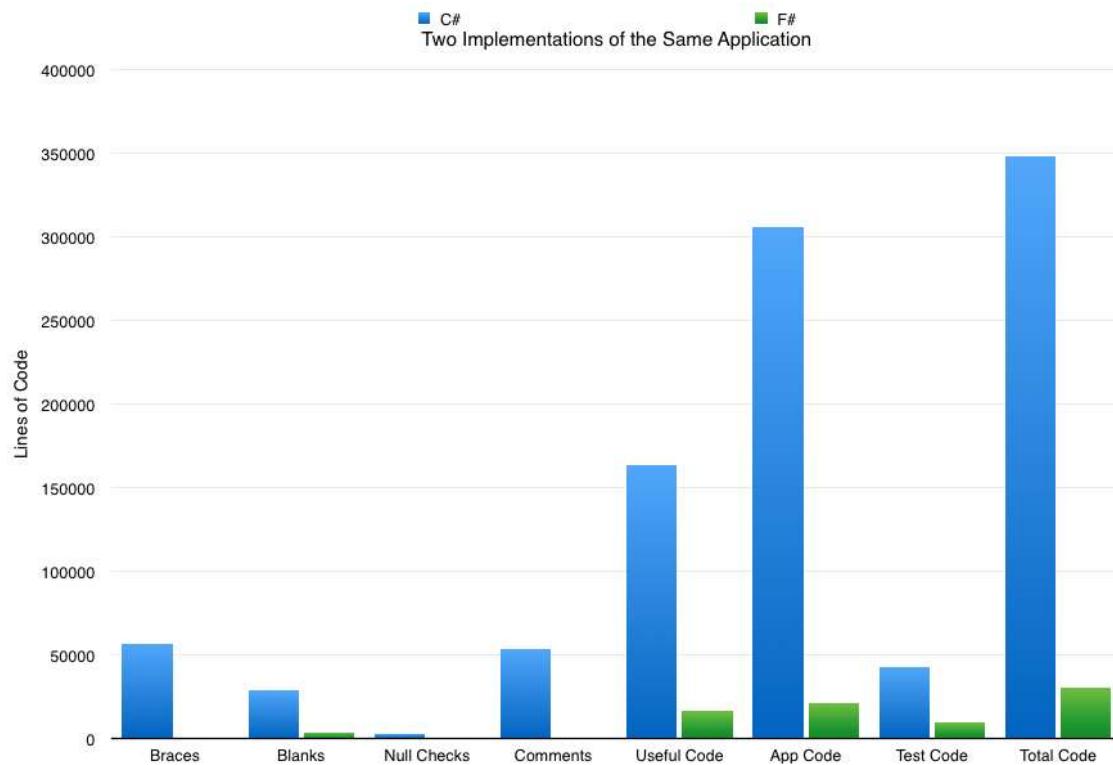
- Peter Norvig

`const :: a -> b -> a`

- Et sterkt, statisk typesystem gir presis spesifisering av lovlig input og output for funksjoner.
- Fanger opp og forhindrer feil allerede i kompileringsfasen
- Debugging i forkant
- Typinferens og generisk programering
- Høyt abstraksjonsnivå

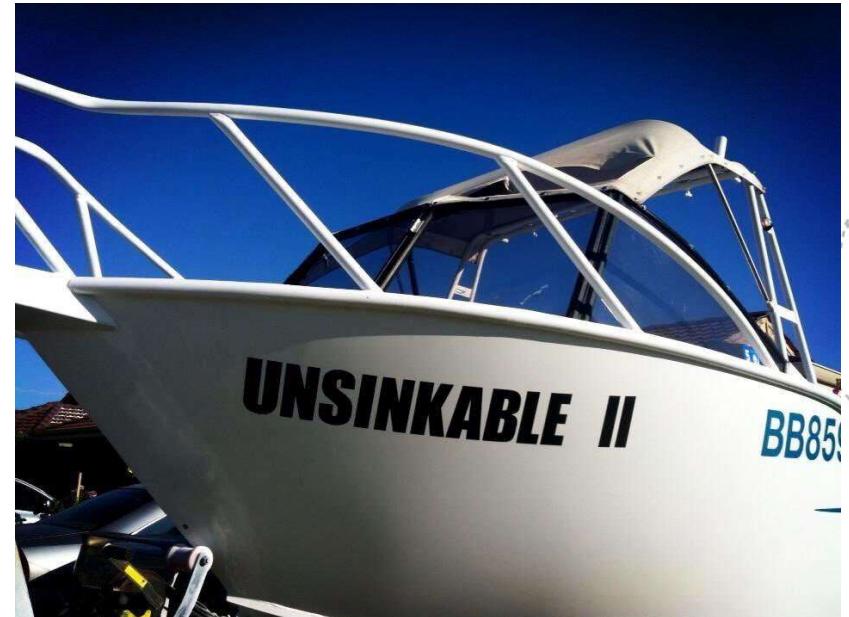


OOP vs FP



F#

- Functional first, full OOP support
- Variables are explicitly declared
- Strong type system
- Algebraic data types
- First class member of .NET
- Excellent tooling
- Easy to learn
- Microsoft
- Open source



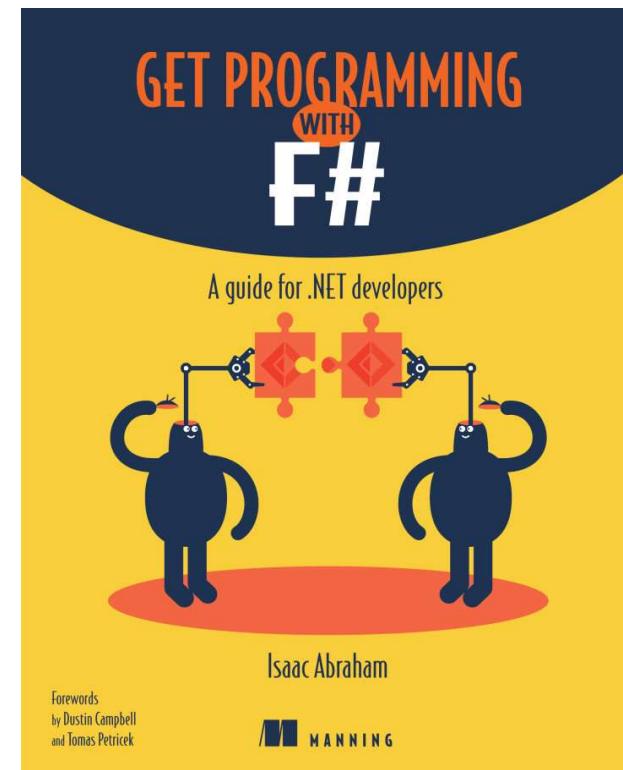
F#

One language for:

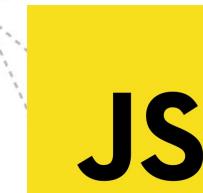
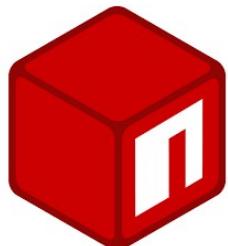
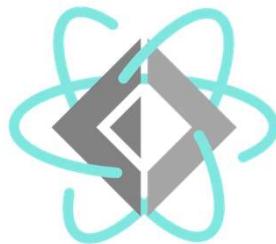
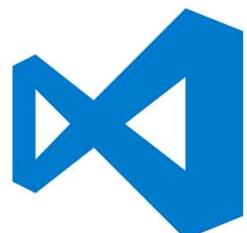
- Server applications
- Client applications
- Mobile applications

F# can be transpiled to JavaScript:

- Fable and WebSharper
- Reactive, asynchronous programming
- Elm and the MVU model



The F# ecosystem



“Simplicity is the prerequisite of reliability”

E. Dijkstra

The goodie bag

- The value of values (Rich Hickey)
- Simple made easy (Rich Hickey)
- F# for Fun and Profit (Scott Wlaschin)
- Stop writing classes (Jack Diederich)
- Beating the averages (Paul Graham)
- Out of the tar pit (Moseley & Marks 2006)

