λ

Matt Might
University of Utah
matt.might.net
@mattmight

Lambda is everywhere!

All you need is lambda.

Lambda multiplies you.

Edward Sapir

Benjamin Whorf

Sapir-Whorf Hypothesis

Edward Sapir

Language limits thought.

Benjamin Whorf

Challenge

Write factorial without using recursion.

Or iteration.

Or conditionals.

Or any numbers.

```
(((lambda f: (((f)((lambda f: ((lambda z: (((f)(((f)(((f)(((f)(((f)
(z)))))))))))))))))))((((((lambda y: ((lambda F: (((F)((lambda x: ((((((y)(y)))
(F)))(x)))))))))))((lambda y: ((lambda F: (((F)((lambda x: ((((((y)(y)))(F)))
(x)))))))))))))((lambda f: ((lambda n: (((((((((((lambda n: (((((n)((lambda _:
((lambda t: ((lambda f: (((f)((lambda void: (void)))))))))))))((lambda t:
((lambda f: (((t)((lambda void: (void)))))))))))))((((((lambda n: ((lambda m:
((((m)((lambda n: ((lambda f: ((lambda z: ((((((n)((lambda g: ((lambda h:
(((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u: (u)))))))))))))(n)))))))
(n)))((lambda f: ((lambda z: (z)))))))))((lambda _: ((((lambda n: (((((n)
((lambda _: ((lambda t: ((lambda f: (((f)((lambda void: (void))))))))))))
((lambda t: ((lambda f: (((t)((lambda void: (void)))))))))))))((((((lambda n:
((lambda m: ((((m)((lambda n: ((lambda f: ((lambda z: ((((((n)((lambda g:
((lambda h: (((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u:
(u)))))))))))))(n)))))))((lambda f: ((lambda z: (z)))))))(n)))))))))((lambda _:
((lambda t: ((lambda f: (((f)((lambda void: (void)))))))))))))((lambda _:
((lambda f: ((lambda z: (((f)(z)))))))))))((lambda _: ((((((lambda n: ((lambda
m: ((lambda f: ((lambda z: (((((m)(((n)(f)))))(z)))))))))))(n)))(((f)
((((((lambda n: ((lambda m: ((((m)((lambda n: ((lambda f: ((lambda z: ((((((n)
((lambda g: ((lambda h: (((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u:
(u)))))))))))))(n)))))))(n)))((lambda f: ((lambda z: (((f)
(z))))))))))))))))))))))))
```

What is λ?

λ is a notation

for anonymous functions

```
$ python
>>> f = lambda x: x + 1

>>> f(3)
4
```

λ is a language

λ is a proof theory

λ is a way of life

Lambda is everywhere!

All you need is lambda.

Lambda multiplies you.

Lambda is everywhere!

`lambda` v_1`,...,`v_n`:` exp

$$\texttt{(lambda (}v_1 \texttt{ ... } v_n\texttt{)}\ exp\texttt{)}$$

$$(\lambda\ (v_1\ \ldots\ v_n)\ exp)$$

$\texttt{function}\ (v_1, \ldots, v_n)\ \ \{\ \texttt{return}\ exp\ ;\ \}$

```
new Procedure () {
 public T run (T1 v1,...,Tn vn) {
   return exp ;
 }
}
```

$$(T_1\ \ v_1, \ldots, T_n\ \ v_n)\ \texttt{->}\ \ exp$$

$$\texttt{function}\ (v_1,\texttt{...},v_n)\ \texttt{return}\ exp\ \texttt{end}$$

$$\texttt{function } (v_1, \ldots, v_n) \texttt{ use } (\mathit{free}) \; \{\texttt{return } \mathit{exp} \;;\}$$

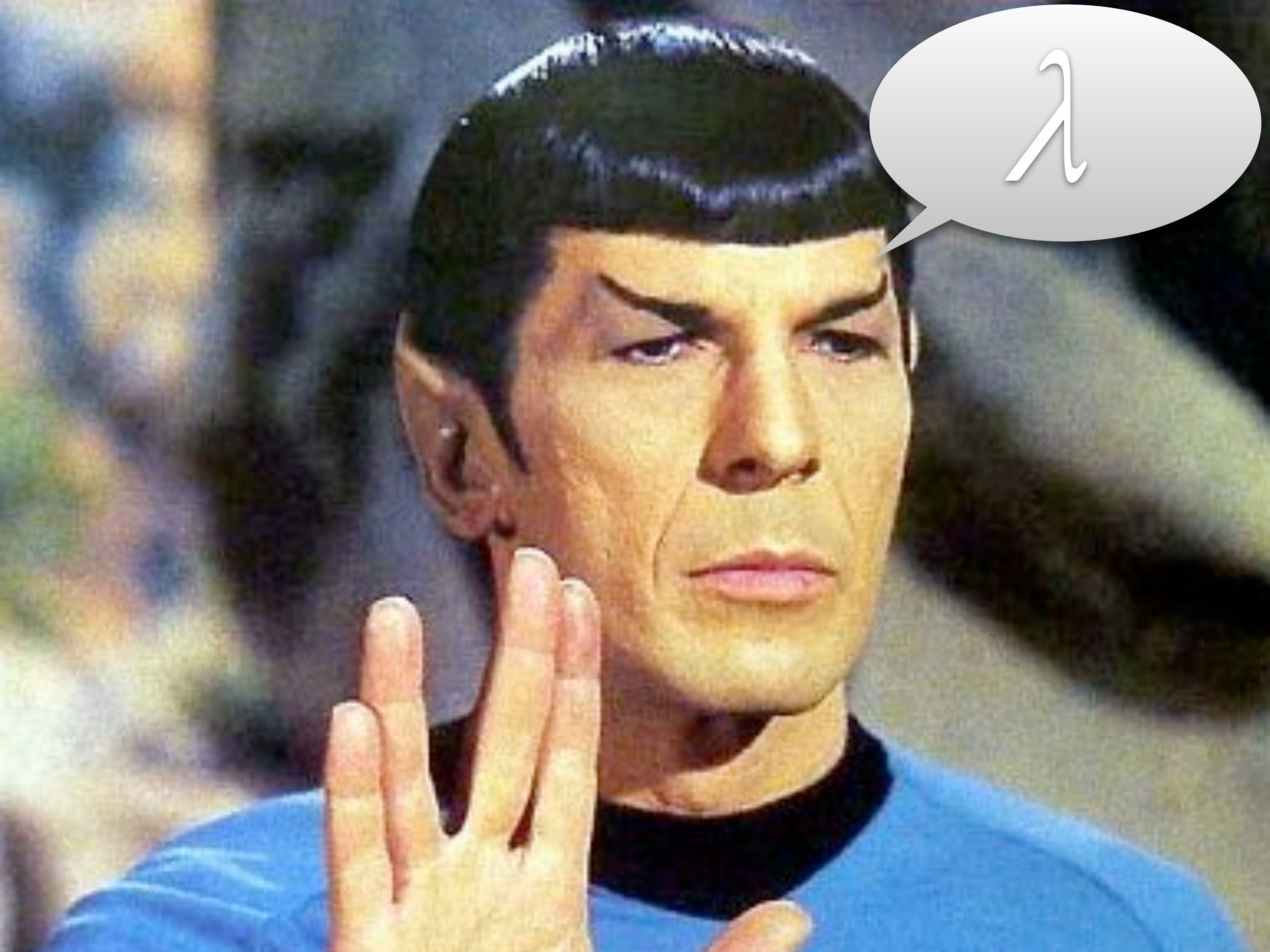
$\texttt{lambda \{|}v_1\texttt{,...,}v_n\texttt{| return } exp \texttt{ \}}$

$$(v_1 \texttt{ : } t_1\texttt{, . . . ,}v_2 \texttt{ : } t_2\texttt{) => } exp$$

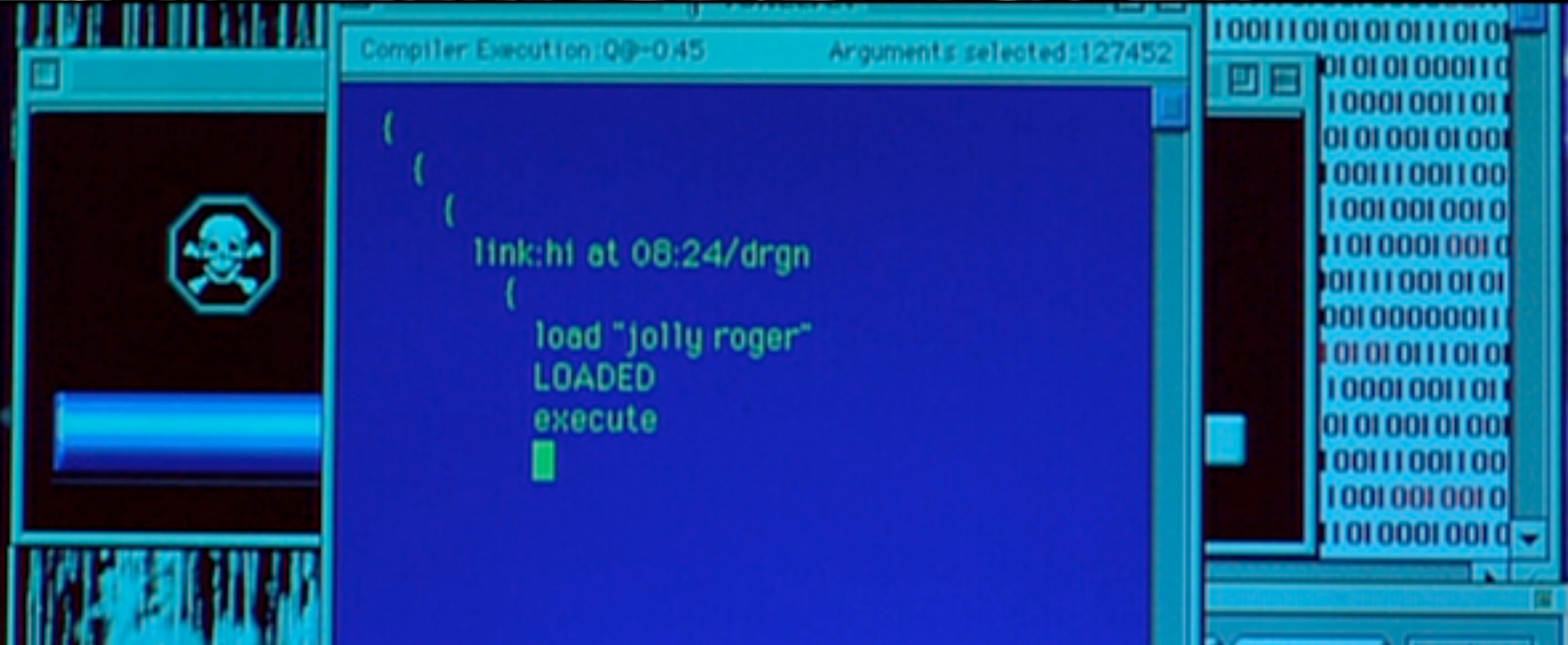
$$[\mathit{free}]\texttt{(}T_1\ v_1\texttt{,}\dots\texttt{,}T_n\ v_n\texttt{)\{ return } \mathit{exp}\texttt{; \}}$$

[] () {}

([] () {}) ()


λ

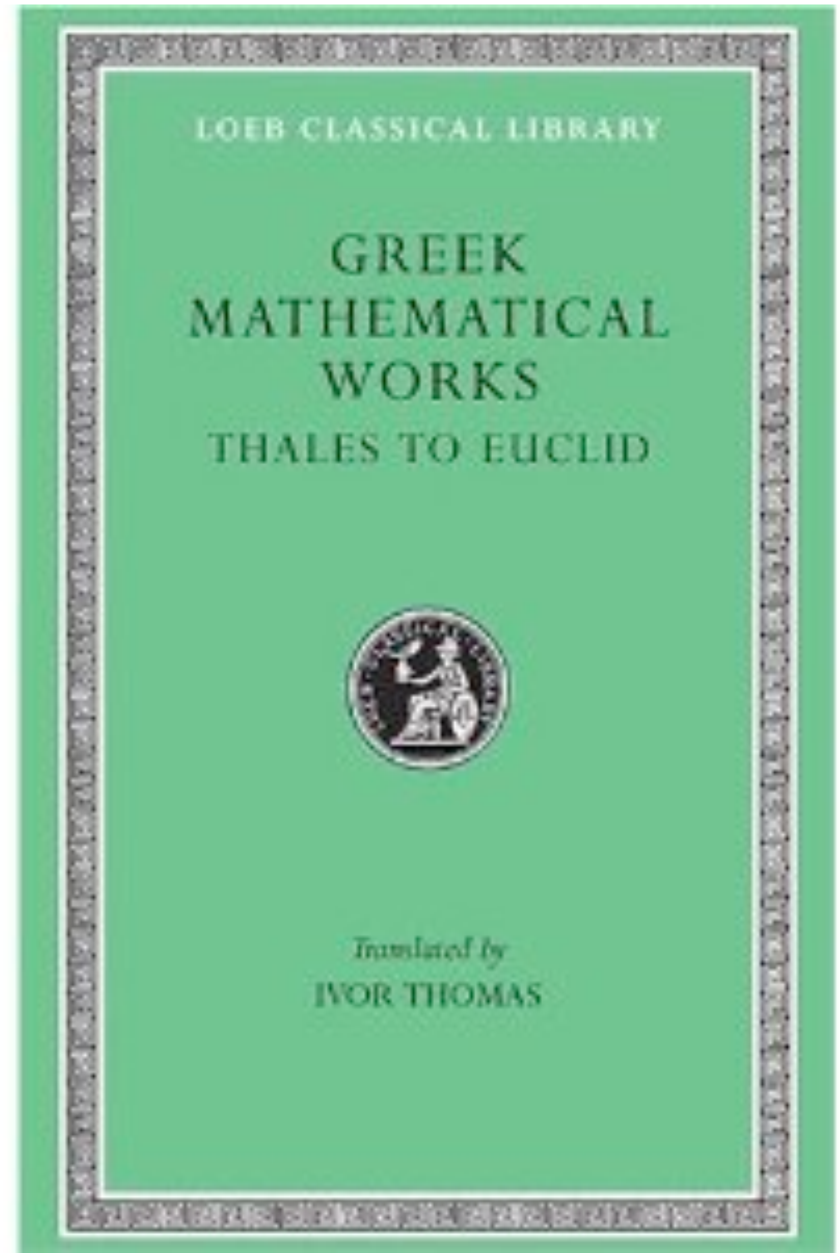

λ


Compiler Execution: Q@-0:45
Arguments selected: 127452
{
{
{
link:hi at 08:24/drgn
{
load "jolly roger"
LOADED
execute

Origins of λ

Origins of notation

(Supposedly) Euclid

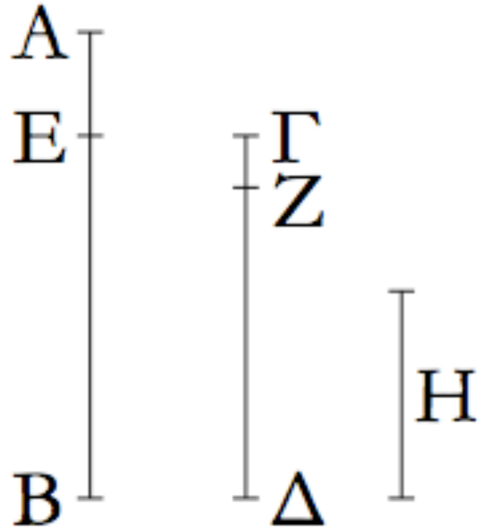


300 BC

2300104 BACD

β΄.

Δύο ἀριθμῶν δοθέντων μὴ πρώτων πρὸς ἀλλήλους τὸ μέγιστον αὐτῶν κοινὸν μέτρον εὑρεῖν.



Ἔστωσαν οἱ δοθέντες δύο ἀριθμοὶ μὴ πρῶτοι πρὸς ἀλλήλους οἱ ΑΒ, ΓΔ. δεῖ δὴ τῶν ΑΒ, ΓΔ τὸ μέγιστον κοινὸν μέτρον εὑρεῖν.

```
gcd(a, b) = (b == 0) ? a : gcd(b, a mod b)
```

Εἰ μὲν οὖν ὁ ΓΔ τὸν ΑΒ μετρεῖ, μετρεῖ δὲ καὶ ἑαυτόν, ὁ ΓΔ ἄρα τῶν ΓΔ, ΑΒ κοινὸν μέτρον ἐστίν. καὶ φανερόν, ὅτι καὶ μέγιστον· οὐδεὶς γὰρ μείζων τοῦ ΓΔ τὸν ΓΔ μετρήσει.

Εἰ δὲ οὐ μετρεῖ ὁ ΓΔ τὸν ΑΒ, τῶν ΑΒ, ΓΔ ἀνθυφαιρουμένου ἀεὶ τοῦ ἐλάσσονος ἀπὸ τοῦ μείζονος λειφθήσεταί τις ἀριθμός, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. μονὰς μὲν γὰρ οὐ λειφθήσεται· εἰ δὲ μή, ἔσονται οἱ ΑΒ, ΓΔ πρῶτοι πρὸς ἀλλήλους· ὅπερ οὐχ ὑπόκειται. λειφθήσεταί τις ἄρα ἀριθμὸς, ὃς μετρήσει τὸν πρὸ ἑαυτοῦ. καὶ ὁ μὲν ΓΔ τὸν ΒΕ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΕΑ, ὁ δὲ ΕΑ τὸν ΔΖ μετρῶν λειπέτω ἑαυτοῦ ἐλάσσονα τὸν ΖΓ, ὁ δὲ ΓΖ τὸν ΑΕ μετρείτω. ἐπεὶ οὖν ὁ ΓΖ τὸν ΑΕ μετρεῖ, ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ, καὶ ὁ ΓΖ ἄρα τὸν ΔΖ μετρήσει. μετρεῖ δὲ καὶ ἑαυτόν· καὶ ὅλον ἄρα τὸν ΓΔ μετρήσει. ὁ δὲ ΓΔ τὸν ΒΕ μετρεῖ· καὶ ὁ ΓΖ ἄρα τὸν ΒΕ μετρεῖ· μετρεῖ δὲ καὶ τὸν ΕΑ· καὶ ὅλον ἄρα τὸν ΒΑ μετρήσει· μετρεῖ δὲ καὶ τὸν ΓΔ· ὁ ΓΖ ἄρα τοὺς ΑΒ, ΓΔ μετρεῖ. ὁ ΓΖ ἄρα τῶν ΑΒ, ΓΔ κοινὸν μέτρον ἐστίν. λέγω δή, ὅτι καὶ μέγιστον. εἰ γὰρ μή ἐστιν ὁ ΓΖ τῶν ΑΒ, ΓΔ μέγιστον κοινὸν μέτρον, μετρήσει τις τοὺς ΑΒ, ΓΔ ἀριθμοὺς ἀριθμὸς μείζων ὢν τοῦ ΓΖ. μετρείτω, καὶ ἔστω ὁ Η. καὶ ἐπεὶ ὁ Η τὸν ΓΔ μετρεῖ, ὁ δὲ ΓΔ τὸν ΒΕ μετρεῖ, καὶ ὁ Η ἄρα τὸν ΒΕ μετρεῖ· μετρεῖ δὲ καὶ ὅλον τὸν ΒΑ· καὶ λοιπὸν ἄρα τὸν ΑΕ μετρήσει. ὁ δὲ ΑΕ τὸν ΔΖ μετρεῖ· καὶ ὁ Η ἄρα τὸν ΔΖ μετρήσει· μετρεῖ δὲ καὶ ὅλον τὸν ΔΓ· καὶ λοιπὸν ἄρα τὸν ΓΖ μετρήσει ὁ μείζων τὸν ἐλάσσονα· ὅπερ ἐστὶν ἀδύνατον· οὐκ ἄρα τοὺς ΑΒ, ΓΔ ἀριθμοὺς ἀριθμός τις μετρήσει μείζων ὢν τοῦ ΓΖ· ὁ ΓΖ ἄρα τῶν ΑΒ, ΓΔ μέγιστόν ἐστι κοινὸν μέτρον [ὅπερ ἔδει δεῖξαι].

Πόρισμα.

Ἐκ δὴ τούτου φανερόν, ὅτι ἐὰν ἀριθμὸς δύο ἀριθμοὺς μετρῇ, καὶ τὸ μέγιστον αὐτῶν κοινὸν μέτρον μετρήσει· ὅπερ ἔδει δεῖξαι.

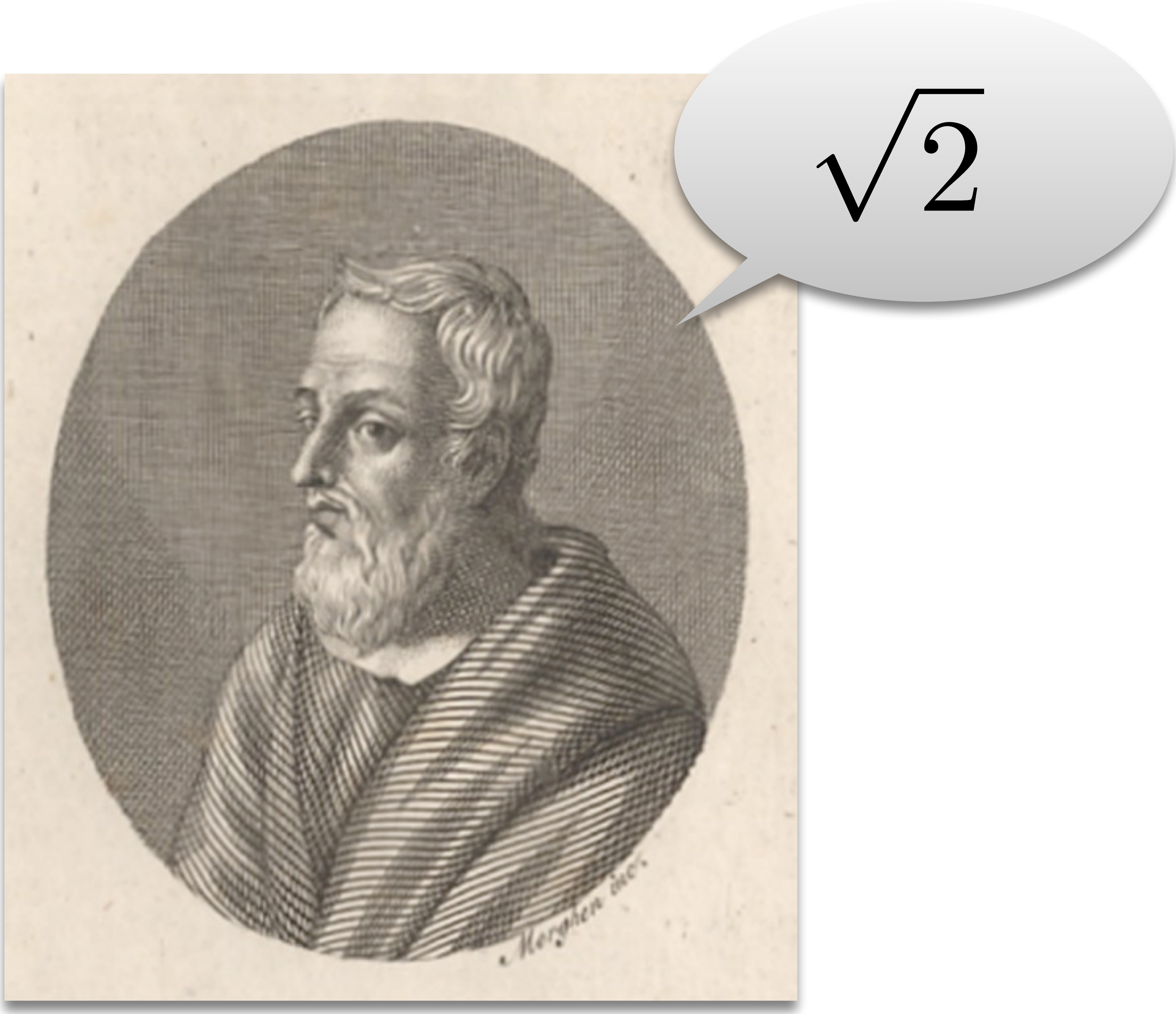
“Limitations”

No zero

No variables/unknowns

No operators

Long division needed Ph.D.

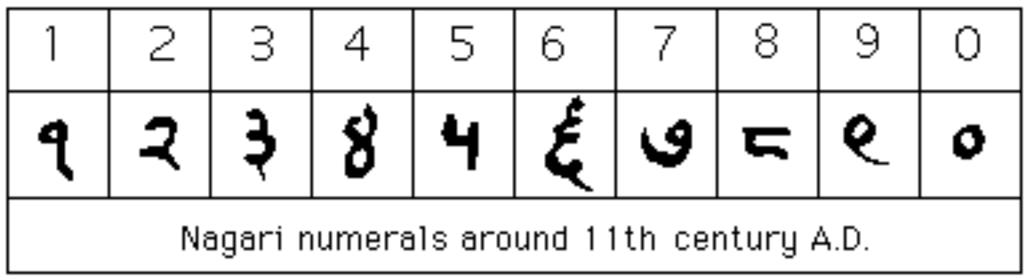


(Supposedly) Hippasus


BURN THE WITCH!

$$1 + 3x = x^2$$

The number such that adding one to three of the root of that number is equal to that number.



Indian numerals (596)

Brahmagupta (?)

The number such that adding one to three of the root of that number is equal to that number.

1	2	3	4	5	6	7	8	9	0
१	२	३	४	५	६	७	८	९	०
Nagari numerals around 11th century A.D.									

Indian numerals (596)

Brahmagupta (?)

The number such that adding 1 to 3 of the root of that number is equal to that number.

et

The number such that adding 1 to 3 of the root of that number is equal to that number.

t

The number such that adding 1 to 3 of the root of that number is equal to that number.

t

The number such that adding 1 to 3 of the root of that number is equal to that number.

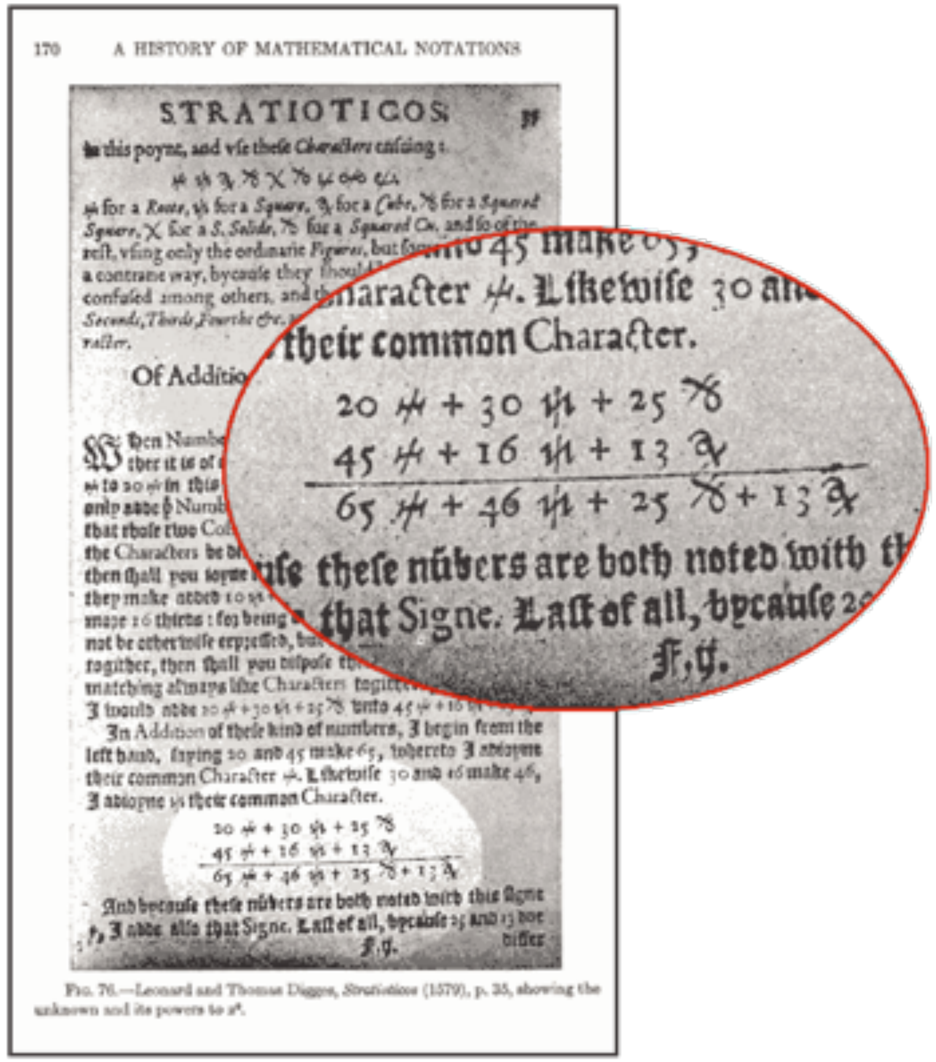
$$+$$

The number such that adding 1 to 3 of the root of that number is equal to that number.

$$+$$

The number such that adding 1 to 3 of the root of that number is equal to that number.

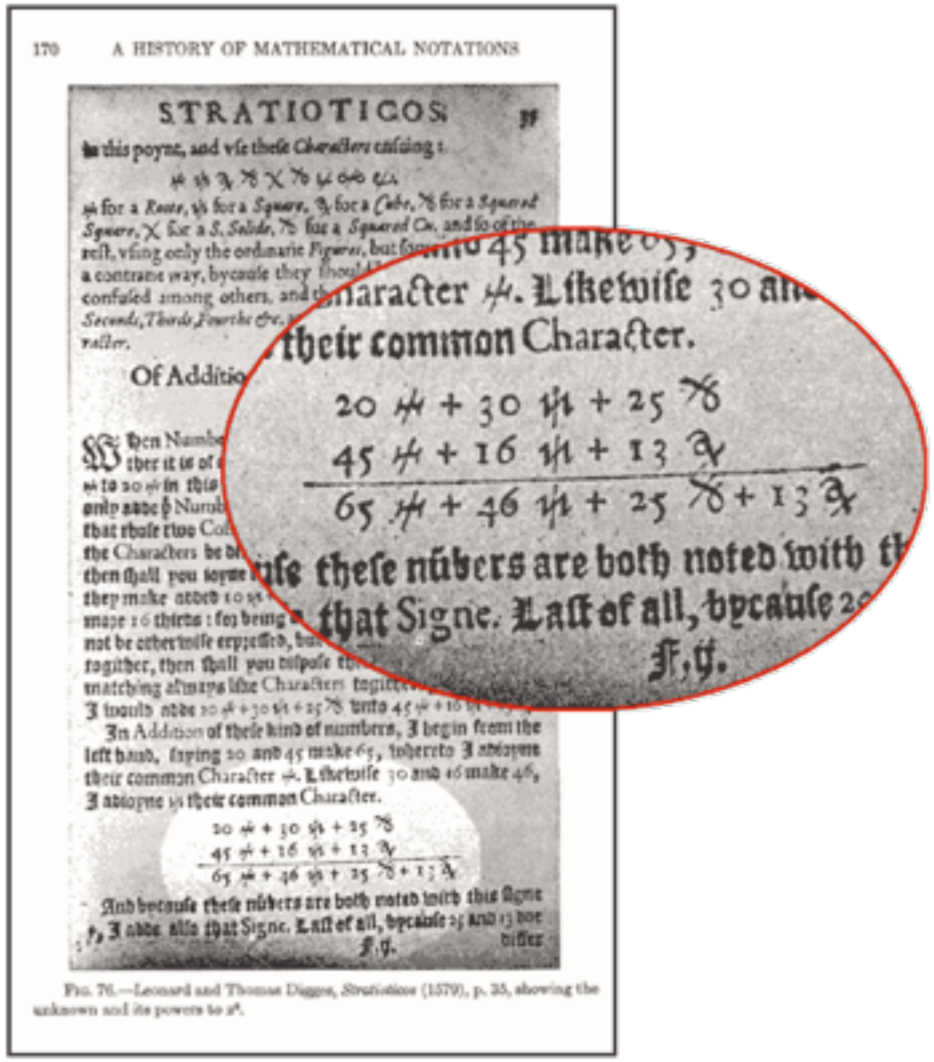
The number such that $1 + 3$ of the root of that number is equal to that number.



Variables (1570)

(Probably) François Viète

The number such that $1 + 3$ of the root of that number is equal to that number.

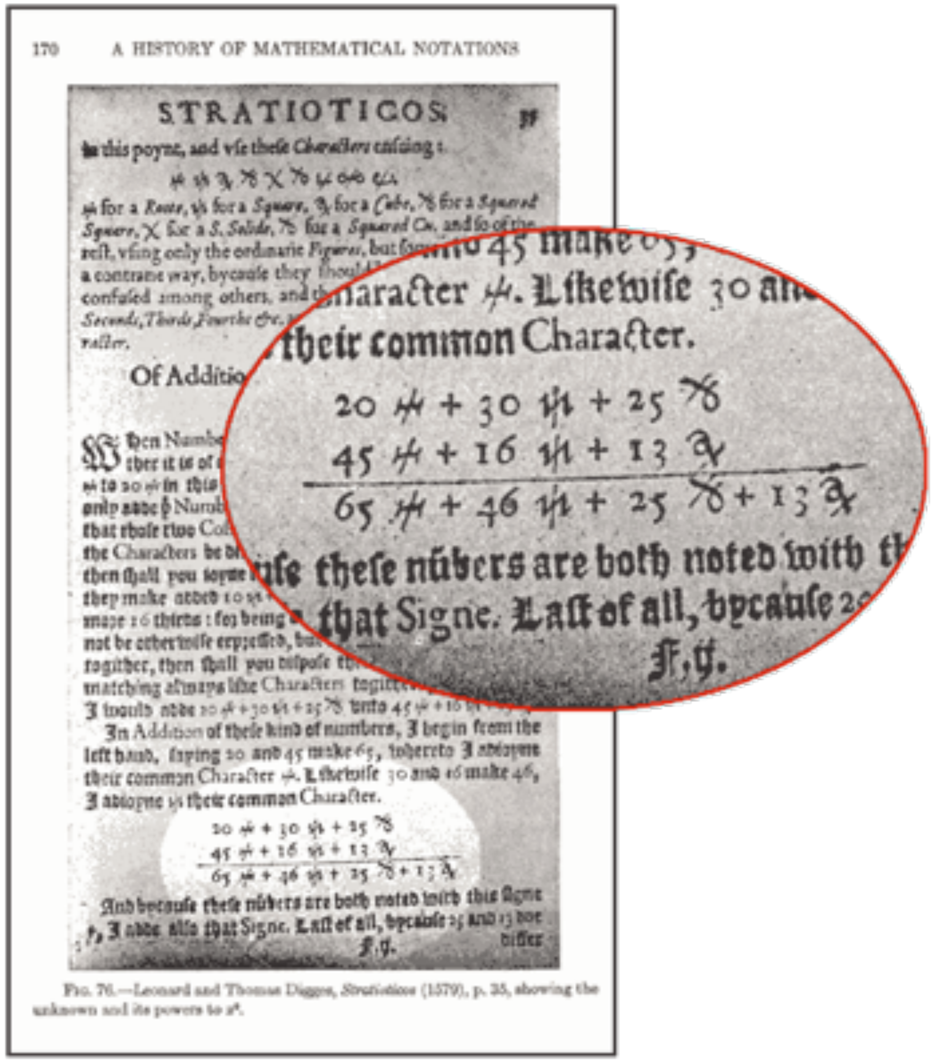


Variables (1570)

(Probably) François Viète

$1 + 3$ ϰ

is equal to ♅



Variables (1570)

(Probably) François Viète

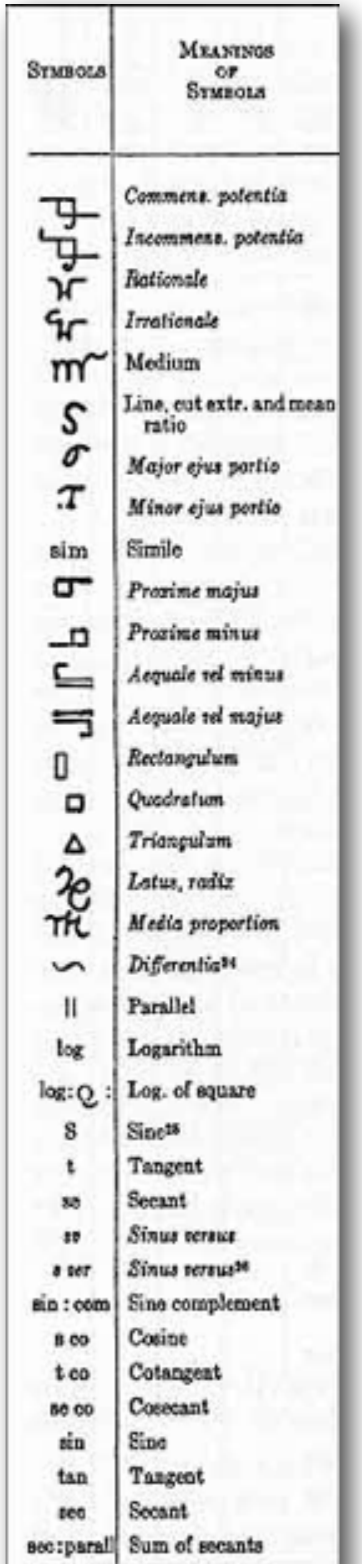
$1 \;+\; 3$ϰ is equal to ♅

$1 + 3$♓ is equal to ♅

William Oughtred

$1 + 3$ϰ is equal to ♅

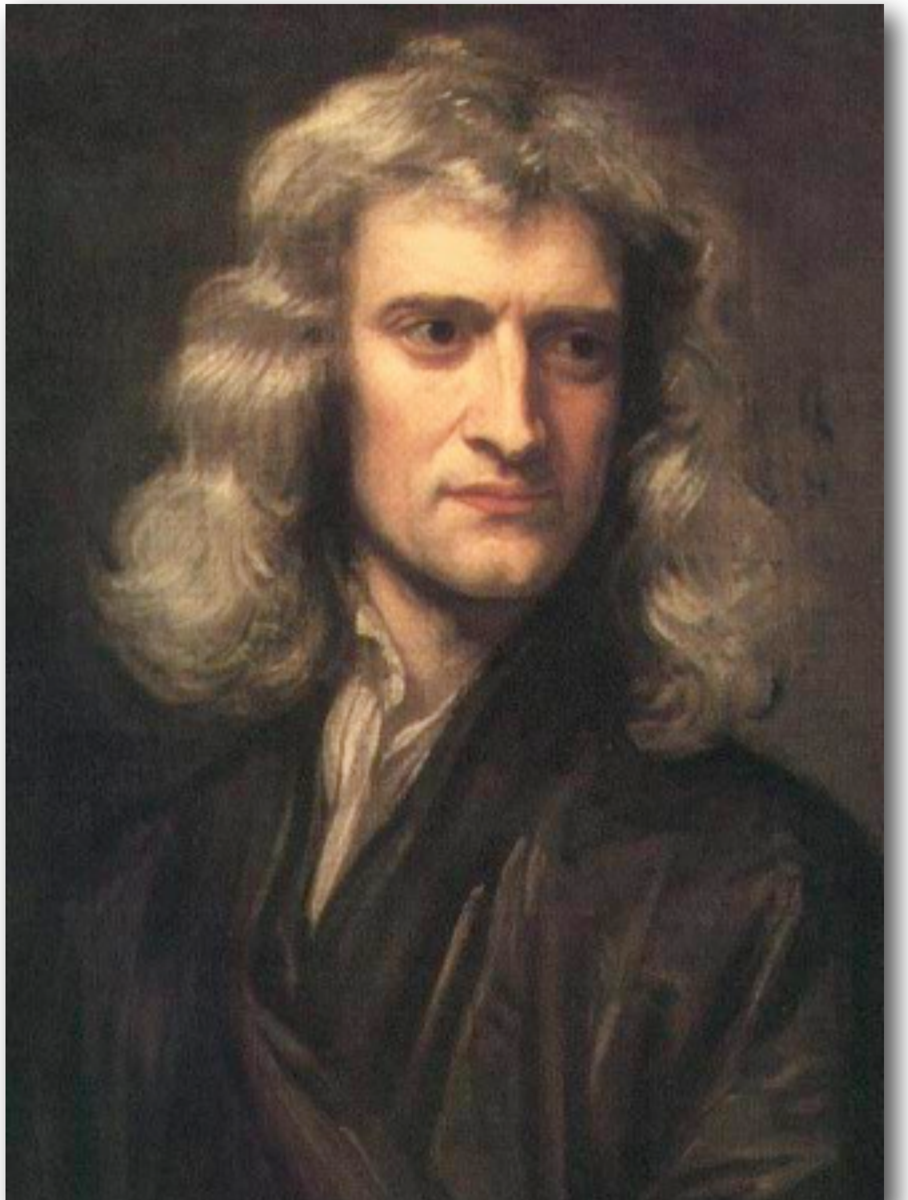
Early 1600s



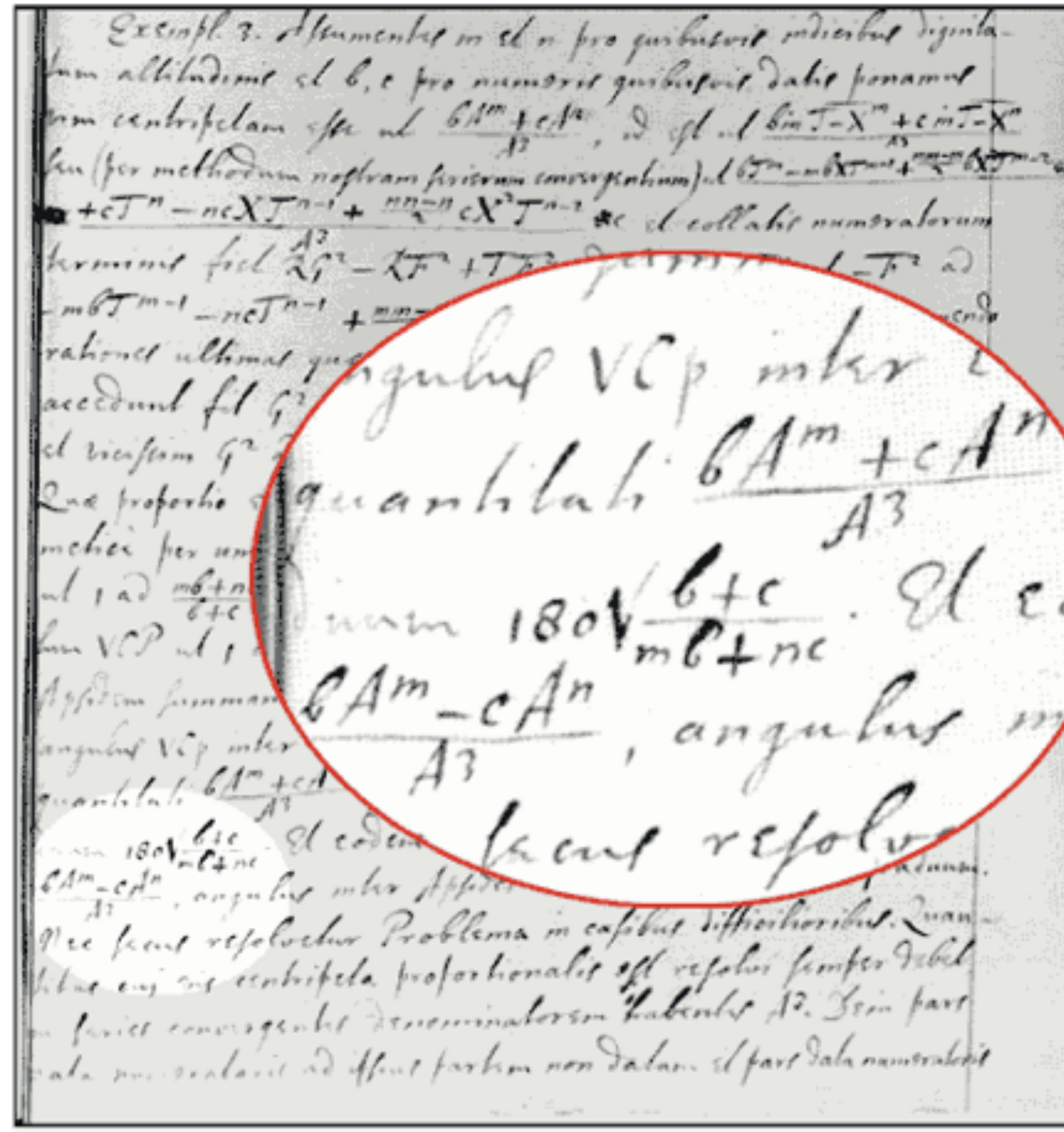
Symbols	Meanings of Symbols
	Commens. potentia
	Incommens. potentia
	Rationale
	Irrationale
	Medium
	Line, cut extr. and mean ratio
	Major ejus portio
	Minor ejus portio
sim	Simile
	Proxime majus
	Proxime minus
	Aequale vel minus
	Aequale vel majus
	Rectangulum
	Quadratum
△	*Triangulum*
	Latus, radix
	Media proportion
∽	*Differentia*[34]
‖	Parallel
log	Logarithm
log:Q:	Log. of square
s	Sine[35]
t	Tangent
se	Secant
s	*Sinus versus*
s ver	*Sinus versus*[36]
sin : com	Sine complement
s co	Cosine
t co	Cotangent
se co	Cosecant
sin	Sine
tan	Tangent
sec	Secant
sec:parall	Sum of secants

William Oughtred

$$1 + 3\varkappa = ♅$$



Isaac Newton



Principia (late 1600s)

$$1 \;+\; 3x = x^2$$

forma generalis autem sumendo $m = 3n$ praebet

$$\frac{\int dx\left(l\frac{1}{x}\right)^{n-1}\cdot\int dx\left(l\frac{1}{x}\right)^{2n-1}}{\int dx\left(l\frac{1}{x}\right)^{4n-1}} = k\int\frac{x^{3nk-1}dx}{(1-x^k)^{1-n}},$$

quibus coniungendis adipiscimur

$$\frac{\left(\int dx\left(l\frac{1}{x}\right)^{n-1}\right)^4}{\int dx\left(l\frac{1}{x}\right)^{4n-1}} = k^3\int\frac{x^{nk-1}dx}{(1-x^k)^{1-n}}\cdot\int\frac{x^{2nk-1}dx}{(1-x^k)^{1-n}}\cdot\int\frac{x^{3nk-1}dx}{(1-x^k)^{1-n}}.$$

Sit nunc $n = \frac{i}{4}$ et sumatur $k = 4$ fietque

$$\frac{\int dx\left(l\frac{1}{x}\right)^{\frac{i}{4}-1}}{\sqrt[4]{1\cdot2\cdot3\cdots(i-1)}} = \sqrt[4]{4^3}\int\frac{x^{i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}\cdot\int\frac{x^{2i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}\cdot\int\frac{x^{3i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}.$$

COROLLARIUM 1

35. Si igitur sit $i = 1$, habebimus

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-3}} = \sqrt[4]{4^3}\int\frac{dx}{\sqrt[4]{(1-x^4)^3}}\cdot\int\frac{x\,dx}{\sqrt[4]{(1-x^4)^3}}\cdot\int\frac{xx\,dx}{\sqrt[4]{(1-x^4)^3}};$$

quae expressio si littera P designetur, erit in genere

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{4n-3}} = \frac{1}{4}\cdot\frac{5}{4}\cdot\frac{9}{4}\cdots\frac{4n-3}{4}P.$$

COROLLARIUM 2

36. Pro altero casu principali sumamus $i = 3$ eritque

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-1}} = \sqrt[4]{2\cdot4^3}\int\frac{x^2dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{x^5dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{x^8dx}{\sqrt[4]{(1-x^4)}}$$

seu facta reductione ad simpliciores formas

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-1}} = \sqrt[4]{8}\int\frac{xx\,dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{x\,dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{dx}{\sqrt[4]{(1-x^4)}};$$

$$1 + 3x = x^2$$

336 EVOLUTIO FORMULAE INTEGRALIS $\int x^{f-1}dx(lx)^{\frac{m}{n}}$ [114–115

forma generalis autem sumendo $m = 3n$ praebet

$$\frac{\int dx\left(l\frac{1}{x}\right)^{n-1}\cdot\int dx\left(l\frac{1}{x}\right)^{3n-1}}{\int dx\left(l\frac{1}{x}\right)^{4n-1}} = k\int\frac{x^{3nk-1}dx}{(1-x^k)^{1-n}},$$

quibus coniungendis adipiscimur

$$\frac{\left(\int dx\left(l\frac{1}{x}\right)^{n-1}\right)^4}{\int dx\left(l\frac{1}{x}\right)^{4n-1}} = k^3\int\frac{x^{nk-1}dx}{(1-x^k)^{1-n}}\cdot\int\frac{x^{2nk-1}dx}{(1-x^k)^{1-n}}\cdot\int\frac{x^{3nk-1}dx}{(1-x^k)^{1-n}}.$$

Sit nunc $n = \frac{i}{4}$ et sumatur $k = 4$ fietque

$$\frac{\int dx\left(l\frac{1}{x}\right)^{\frac{i}{4}-1}}{\sqrt[4]{1\cdot2\cdot3\cdots(i-1)}} = \sqrt[4]{4^3}\int\frac{x^{i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}\cdot\int\frac{x^{2i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}\cdot\int\frac{x^{3i-1}dx}{\sqrt[4]{(1-x^4)^{4-i}}}.$$

COROLLARIUM 1

35. Si igitur sit $i = 1$, habebimus

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-3}} = \sqrt[4]{4^3}\int\frac{dx}{\sqrt[4]{(1-x^4)^3}}\cdot\int\frac{xdx}{\sqrt[4]{(1-x^4)^3}}\cdot\int\frac{xxdx}{\sqrt[4]{(1-x^4)^3}};$$

quae expressio si littera P designetur, erit in genere

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{4n-3}} = \frac{1}{4}\cdot\frac{5}{4}\cdot\frac{9}{4}\cdots\frac{4n-3}{4}P.$$

COROLLARIUM 2

36. Pro altero casu principali sumamus $i = 3$ eritque

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-1}} = \sqrt[4]{2\cdot4^3}\int\frac{x^2dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{x^5dx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{x^8dx}{\sqrt[4]{(1-x^4)}}$$

seu facta reductione ad simpliciores formas

$$\int dx\sqrt[4]{\left(l\frac{1}{x}\right)^{-1}} = \sqrt[4]{8}\int\frac{xxdx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{xdx}{\sqrt[4]{(1-x^4)}}\cdot\int\frac{dx}{\sqrt[4]{(1-x^4)}};$$

$$f(x) = 1 + 3x - x^2$$

$$f(x)$$

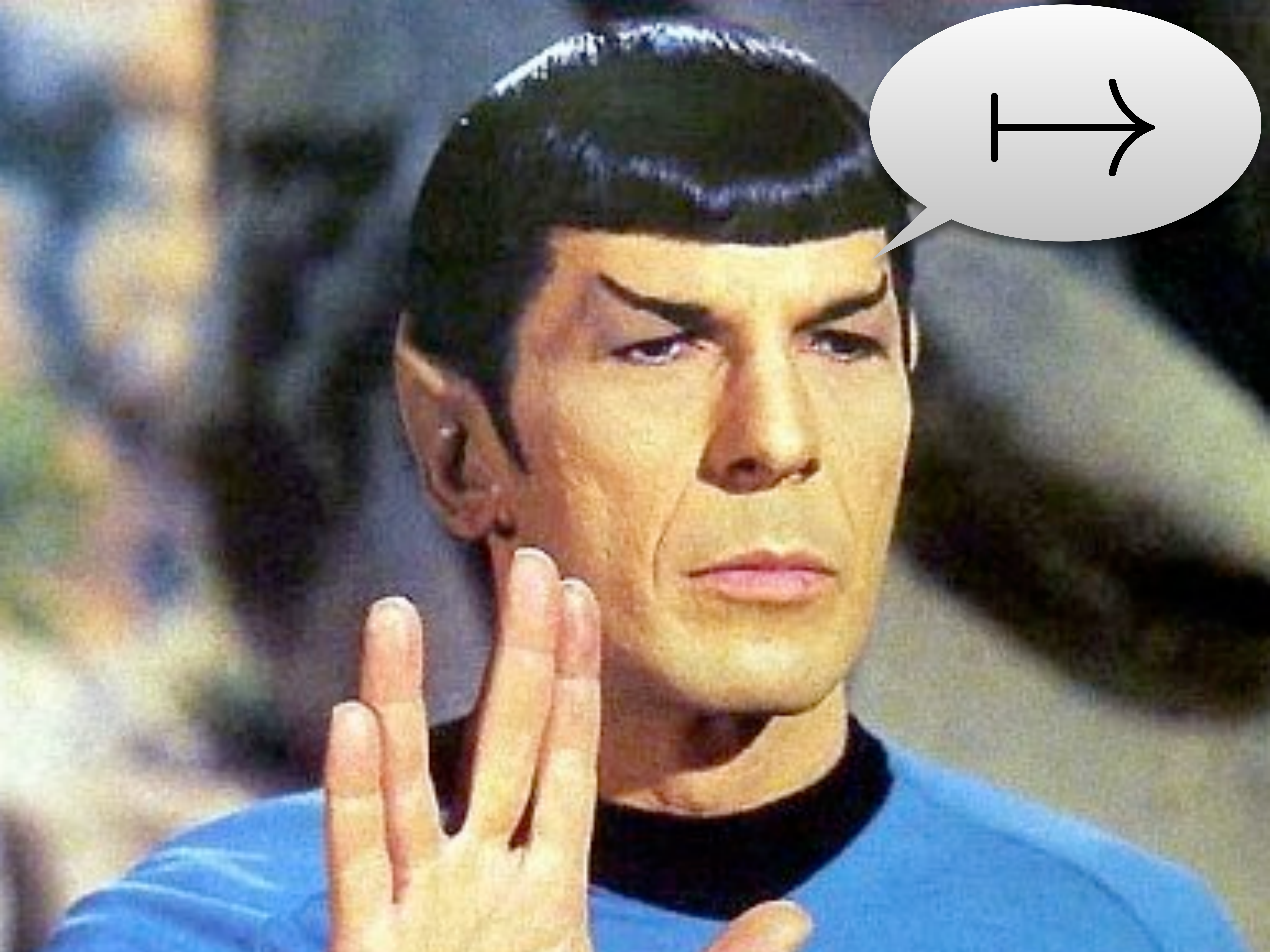
$$f$$

$$f^2$$

f

$$x \mapsto x^2 + 3$$

```
lambda x: x*x + 3
```


$\mapsto$

What else is out there?

Giuseppe Peano (1800s)

$$\{\ \}$$

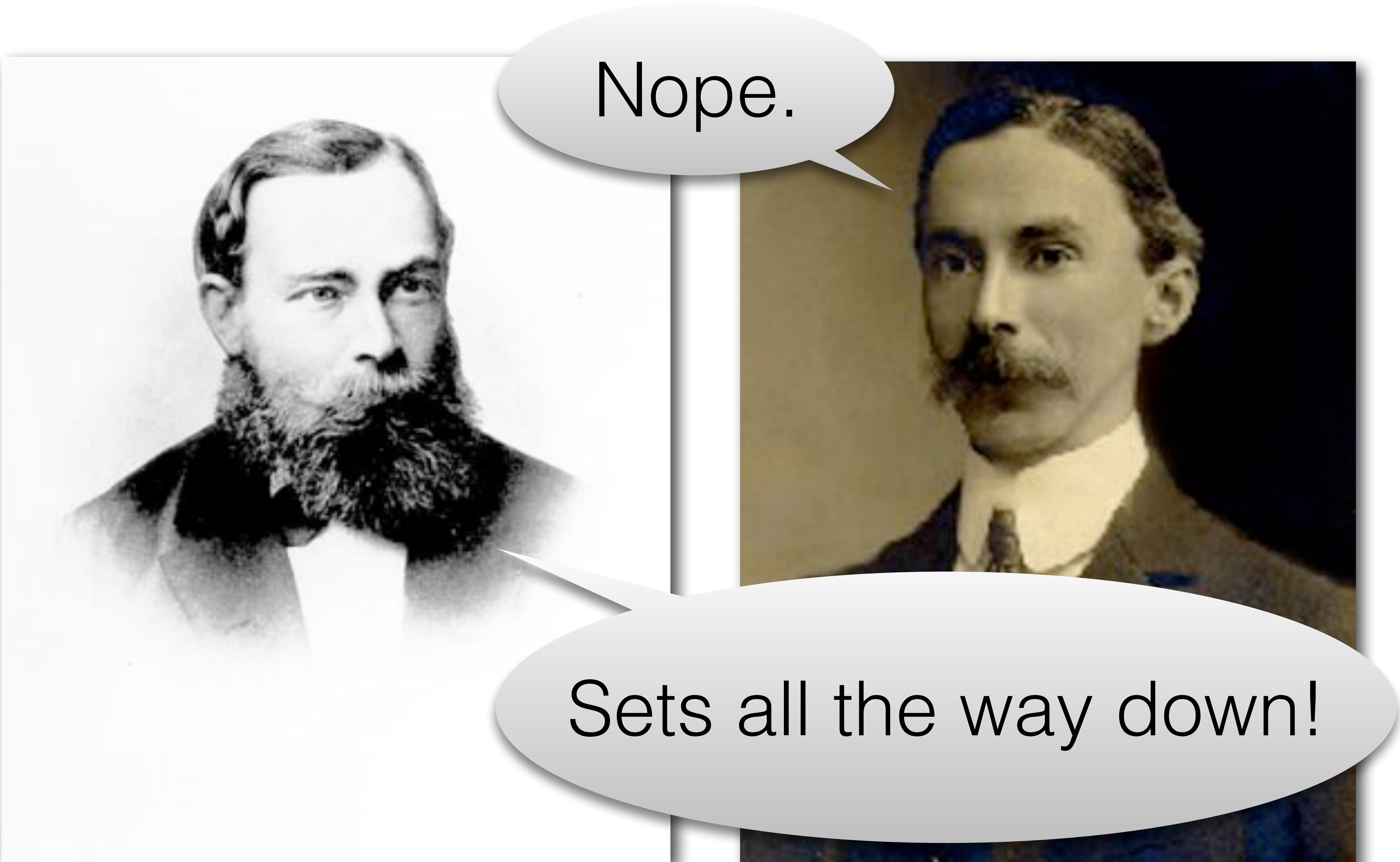
$$\{1, 2\}$$

$$\left\{ \left\{ \quad \right\} \right\}$$

Unification of math & logic

Every integer greater than two can be written as the sum of two primes.

$$\forall n : (n > 2) \implies \exists a, b : p(a) \wedge p(b) \wedge a + b = n$$



Gottlob Frege

Bertrand Russell

Alonzo Church

(My great^5 grand advisor!)

Why not functions?

$$f(x) = x^2 - 4$$

$$f = \lambda x . x^2 - 4$$

$$\lambda x.\, x^2 - 4$$

$$(\lambda x . x^2 - 4)(2)$$

0

λ-calculus

v

$$e_1(e_2)$$

$$\lambda v.e$$

$$v \mid \lambda v.e \mid e_1(e_2)$$

$$v$$

$$\lambda v.e$$

$$e_1(e_2)$$

v

`lambda` v`:` e

e_1`(`e_2`)`

Lambda is everywhere!

All you need is lambda.

Lambda multiplies you.

All you need is lambda.

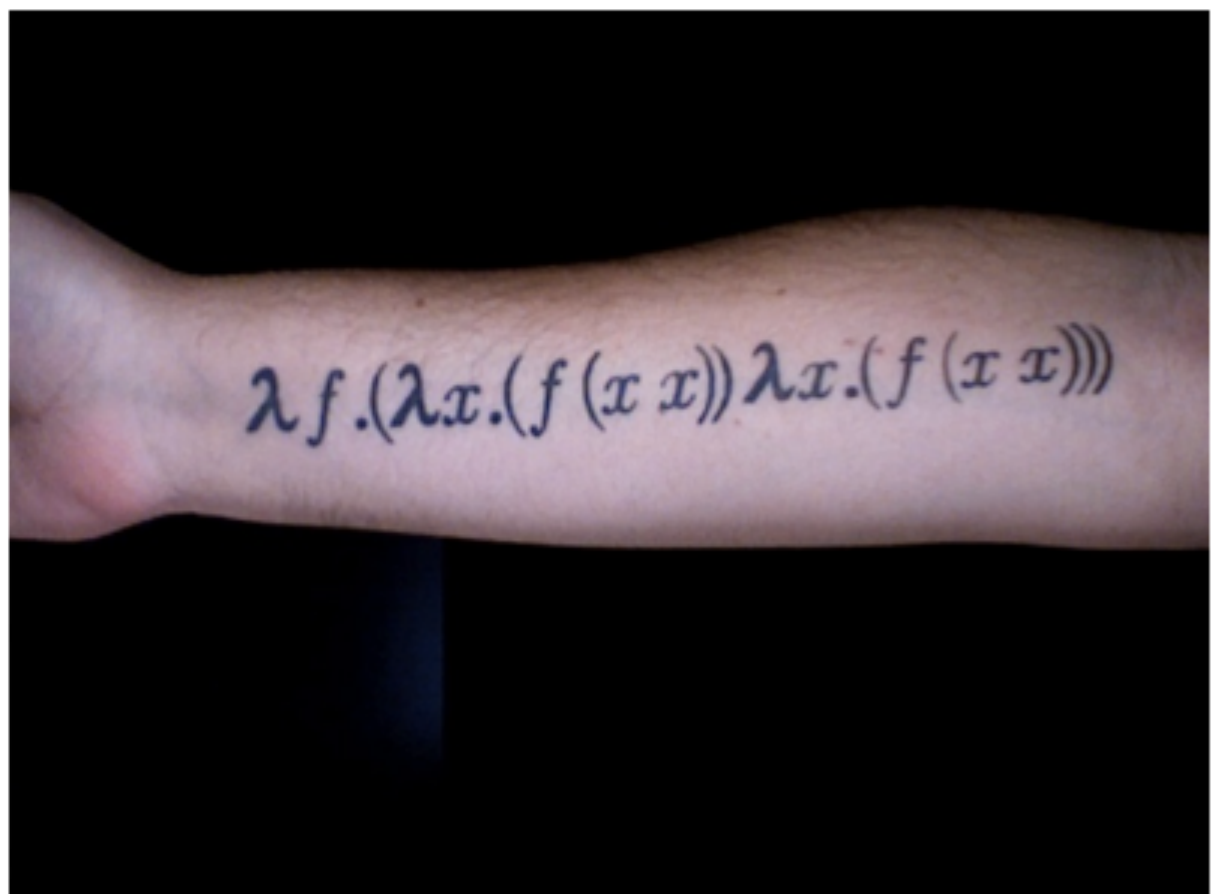
`lambda => language`

Compiling to lambda-calculus: Turtles all the way down

[article index] [email me] [@mattmight] [+mattmight] [rss]

My compilers class always starts with a full lecture on the lambda-calculus.

It leaves behind only the dedicated.



Barendregt is a helluva drug.

The lambda-calculus is a minimal programming language.

Multi-argument functions

f(~~x~~,y)

`lambda` x`:` e

`lambda` x , y : e

lambda x: lambda y : e

$$f(x, y)$$

$$f(x)(y)$$

```
$ python
>>> f = lambda x,y: x + y

>>> g = lambda x: lambda y: x + y

>>> f(1,2)
3

>>> g(1)(2)
3
```

Booleans

$\texttt{IF}\ (\mathit{cond})\ (\mathit{true})\ (\mathit{false})$

$\texttt{IF (TRUE)}\ (\mathit{true})\ (\mathit{false})$

true

$\mathtt{IF}\ (\mathtt{FALSE})\ (\mathit{true})\ (\mathit{false})$

false

IF ($cond$) ($true$) ($false$)

$$cond\ (true)\ (false)$$

$\texttt{IF}\ (\textit{cond})\ (\textit{true})\ (\textit{false})$

```
IF = (lambda c :
        lambda t :
         lambda f :
          c(t)(f))
```

```
TRUE = (lambda t:
          lambda f:
            t)
```

```
FALSE = (lambda t:
           lambda f:
             f)
```

Natural numbers

What is a natural?

Zero

Another natural + 1

How do we use naturals?

Iteration!

$$\texttt{ZERO}\ (f)\ (z)$$

$$\mathtt{ZERO}\ (f)\ (z) == z$$

$\texttt{ONE}\ (f)\ (z)$

$$\texttt{ONE}\ (f)\ (z) == f(z)$$

$$\mathtt{TWO}\ (f)\ (z) == f(f(z))$$

$$\mathtt{THREE}\ (f)\ (z) == f(f(f(z)))$$

$$\mathtt{ZERO}\ (f)\ (z) == z$$

```
ZERO = (lambda f :
          lambda z :
           f(z))
```

SUCC(n)

$$\mathtt{SUCC(n)(f)(z)\ =\ f(n(f)(z))}$$

```
SUCC =   n  f  z    f(n(f)(z))
```

```
SUCC = (
    lambda n:
      lambda f:
        lambda z: f(n(f)(z)))
```

```
SUM = (lambda a:
         lambda b:
          lambda f:
           lambda z:
            a(f)(b(f)(z)))
```

```
MUL = (lambda a:
        lambda b:
         lambda f:
          lambda z:
  a(lambda z:b(f)(z))(z))
```

Pairs

PAIR (a) (b)

$$\texttt{LEFT (PAIR (}a\texttt{) (}b\texttt{))} = a$$

$$\mathtt{RIGHT}\ (\mathtt{PAIR}\ (a)\ (b)) = b$$

```
PAIR = (lambda a:
          lambda b:
           lambda f: f(a)(b))
```

```
RIGHT = (lambda p:
   p(lambda l: lambda r: r))
```

```
LEFT  = (lambda p:
   p(lambda l: lambda r: l))
```

Lists

NIL

$$\texttt{CONS}\ (hd)\ (tl)$$

$$\mathtt{HEAD}\ (\mathtt{CONS}\ (hd)\ (tl)) = hd$$

$$\mathtt{TAIL}\ (\mathtt{CONS}\ (hd)\ (tl)) = tl$$

$$\mathtt{CONSP}\ (\mathtt{CONS}\ (hd)\ (tl)) = \mathtt{TRUE}$$

$$\texttt{NILP (CONS (}hd\texttt{) (}tl\texttt{)) = FALSE}$$

NILP (NIL) = TRUE

CONSP (NIL) = FALSE

```
VOID = lambda _:_

NIL = lambda oncons: lambda onnil: onnil(VOID)

CONS = (lambda hd: lambda tl:
         lambda oncons: lambda onnil:
          oncons(hd)(tl))

CONSP = lambda l: l (lambda hd: lambda tl: TRUE) (lambda void: FALSE)

NILP = lambda l: l (lambda hd: lambda tl: FALSE) (lambda void: TRUE)

HEAD = lambda l: l (lambda hd: lambda tl: hd) (VOID)

TAIL = lambda l: l (lambda hd: lambda tl: tl) (VOID)
```

Recursion

Non-termination

```
(lambda f:f(f))(lambda g:g(g))
```

```
U = lambda f: f(f)
```

U(U)

Recursion is self-reference

self-application

=>

self-reference

$$f = \ldots f \ldots$$

$$f = \ \ldots\ \mathtt{U(h)}\ \ldots$$

f = `U(lambda h: ... U(h) ...)`

```
f = U(lambda h: ... U(h) ...)
```

```
fact = U(lambda h:
     lambda n: 1 if n <= 0 else n * (U(h))(n-1))
```

We can do better!

```
fact = U(lambda h:
     lambda n: 1 if n <= 0 else n * (U(h))(n-1))
```

```
fact = Y(lambda f:
     lambda n: 1 if n <= 0 else n *      f(n-1))
```

Define recursive function…

…as fixed point…

…of non-recursive function.

Define fixed point finder…

…without recursion.

x is a **fixed point** of F if $F(x) = x$.

$$\mathrm{Y(F)} = \mathrm{x} \quad \text{such that} \quad \mathrm{x} = \mathrm{F(x)}$$

$$Y(F) = x \quad \text{such that} \quad x = F(Y(F))$$

$$Y(F) = F(Y(F))$$

```
Y = lambda F:F(Y(F))
```

```
Y = lambda F: F(lambda x: Y(F)(x))
```

```
Y = U(lambda h: lambda F: F(lambda x: U(h)(F)(x)))
```

```
Y = U(lambda h: lambda F: F(lambda x: h(h)(F)(x)))
```

```
Y = ((lambda h:lambda F:F(lambda x:h(h)(F)(x)))
     (lambda h:lambda F:F(lambda x:h(h)(F)(x))))
```

```
$ python
>>> Y = ((lambda h: lambda F: F(lambda x: h(h)(F)(x)))
         (lambda h: lambda F: F(lambda x: h(h)(F)(x))))

>>> fact = Y(lambda f: lambda n: 1 if n <= 0 else n * f(n-1))

>>> fact(8)
40320
```

```
(((lambda f: (((f)((lambda f: ((lambda z: (((f)(((f)(((f)(((f)(((f)
(z)))))))))))))))))))((((((lambda y: ((lambda F: (((F)((lambda x: (((((((y)(y)))
(F)))(x)))))))))))((lambda y: ((lambda F: (((F)((lambda x: (((((((y)(y)))(F)))
(x)))))))))))))((lambda f: ((lambda n: ((((((((((((lambda n: (((((n)((lambda _:
((lambda t: ((lambda f: (((f)((lambda void: (void)))))))))))))((lambda t:
((lambda f: (((t)((lambda void: (void)))))))))))))((((((lambda n: ((lambda m:
(((((m)((lambda n: ((lambda f: ((lambda z: (((((((n)((lambda g: ((lambda h:
(((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u: (u)))))))))))))(n)))))))
(n)))((lambda f: ((lambda z: (z)))))))))((lambda _: (((lambda n: (((((n)
((lambda _: ((lambda t: ((lambda f: (((f)((lambda void: (void)))))))))))))
((lambda t: ((lambda f: (((t)((lambda void: (void)))))))))))))((((((lambda n:
((lambda m: (((((m)((lambda n: ((lambda f: ((lambda z: (((((((n)((lambda g:
((lambda h: (((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u:
(u)))))))))))))(n)))))))((lambda f: ((lambda z: (z)))))))(n)))))))))((lambda _:
((lambda t: ((lambda f: (((f)((lambda void: (void)))))))))))))((lambda _:
((lambda f: ((lambda z: (((f)(z)))))))))))((lambda _: ((((((lambda n: ((lambda
m: ((lambda f: ((lambda z: (((((m)(((n)(f)))))(z)))))))))))(n)))(((f)
((((((lambda n: ((lambda m: (((((m)((lambda n: ((lambda f: ((lambda z: (((((((n)
((lambda g: ((lambda h: (((h)(((g)(f)))))))))))((lambda u: (z)))))((lambda u:
(u)))))))))))))(n)))))))(n)))((lambda f: ((lambda z: (((f)
(z))))))))))))))))))))))))
```

Fixed-point combinators in JavaScript: Memoizing recursive functions

It comes as a surprise to many programmers that it is possible to express a "recursive" function like factorial without using recursion or iteration.

The technique involved is subtle but powerful: the recursive function is computed as the "fixed point" of a non-recursive function. To compute the fixed point, we can use the **Y combinator**, which is itself a non-recursive function that computes fixed points.

> *That this manages to work is truly remarkable.*
>
> --Sussman and Steele on the Y Combinator

As a practical application of this theory, recursive functions expressed as fixed points allow the use of a memoizing fixed-point combinator. The combinator approach to recursion makes it possible to cache the internal calls to a recursive function automatically.

For example, this caching turns the naive, exponential implementation of Fibonacci into the optimized, linear-time version *for free*.

Read below to see how to do this all of this in JavaScript, courtesy of its anonymous function construct.

Yacc is dead: An update

[article index] [email me] [@mattmight] [+mattmight] [rss]

The draft of "Yacc is Dead" that David Darais and I posted on arXiv received some attention, so we'd like to share updates and feedback.

(You don't have to read the draft to read this post; it's self-contained.)

source: wallpaper-9.com

The draft claims that Brzozowski's derivative eases parser implementation; that it handles *all* CFGs (yes, including left-recursive); that it seems to be efficient in practice; and that it should eventually be efficient in theory too.

We reply to Russ Cox's "Yacc is Not Dead" by running the example that he

Lambda multiplies you.

λ is a gateway drug

Higher-order programming (map, fold, reduce, …)

Functional programming

Laziness

Monads → relational programming

Continuation-passing style → **nodejs**

Continuations → time travel, logic programming

Type theory

Curry-Howard isomorphism → dependent types

→ theorem proving

Thanks!

matt.might.net

```
TRUE = (lambda t:
          lambda f:
            t)
```

```
TRUE = (lambda t:
          lambda f:
            t())
```

```
FALSE = (lambda t:
           lambda f:
             f)
```

```
FALSE = (lambda t:
             lambda f:
                 f())
```

```
IF (cond)
   (lambda: true)
   (lambda: false)
```


Print Screen SysRq
Scroll Lock
Pause Break
F12
Insert
Home
Page Up
Delete
End
Page Down
Num Lock
Enter
Ctrl


TITAN


MAGNETOM Skyra
SIEMENS
Welcome