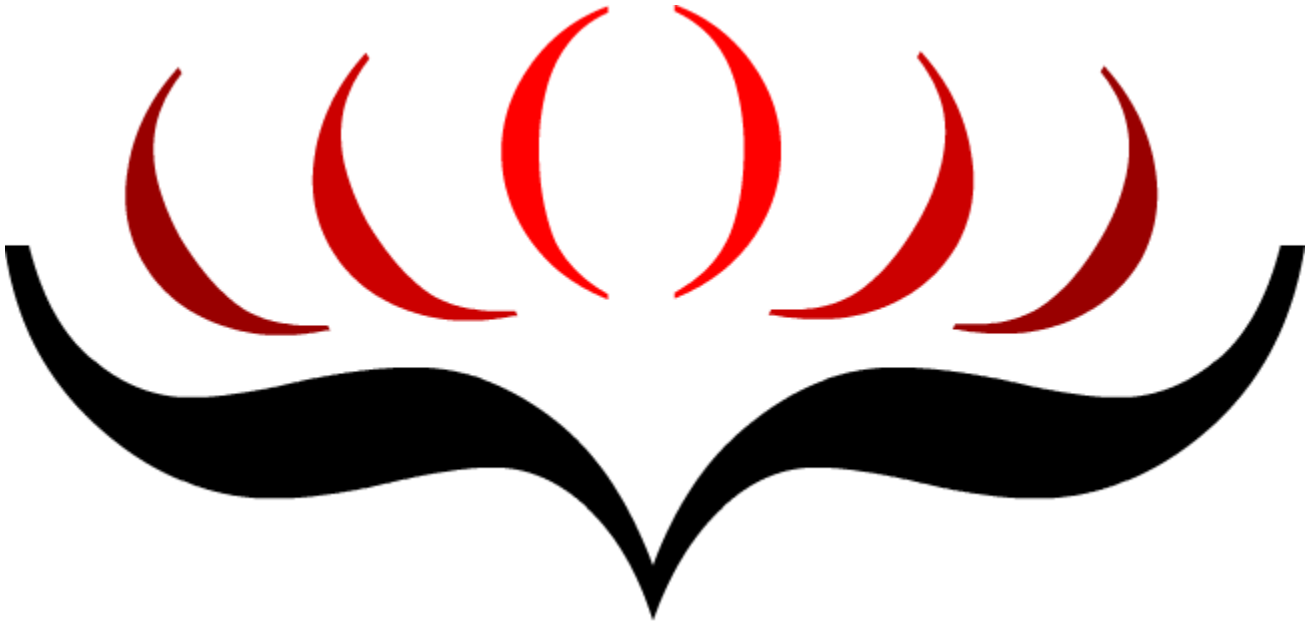
The philosophy and history of functional programming



Uladzimir Aliakseichanka

Agenda

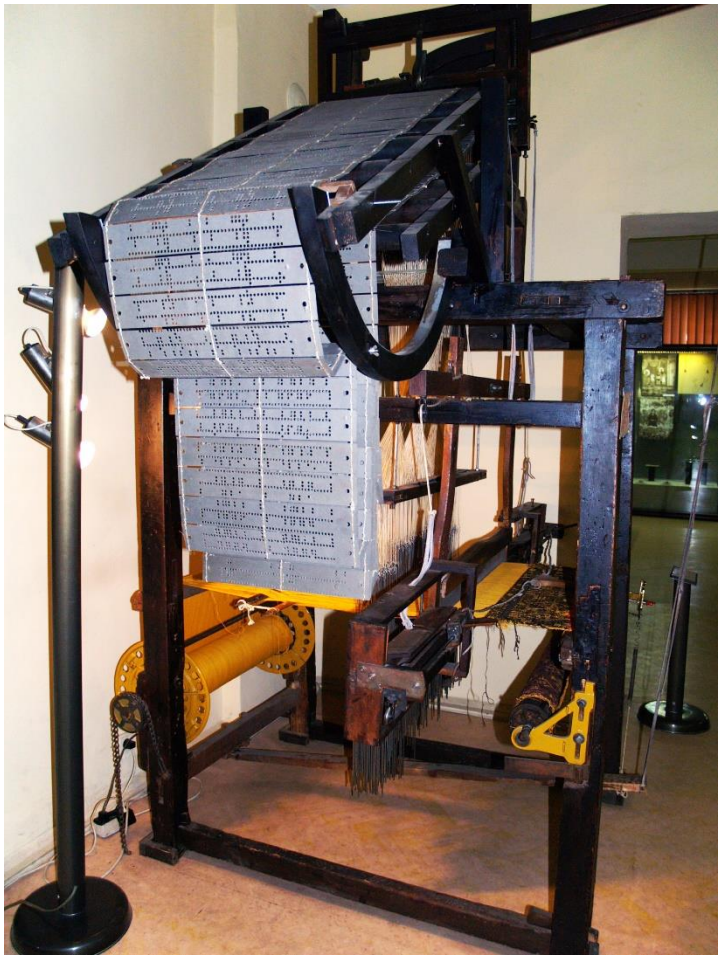
- Joseph Jacquard and Jacquard loom
- The theoretical basis of imperative programming
- The theoretical basis functional programming
- The first (electric) computer
- The LISP and other languages

Historical origins

Joseph Marie Jacquard

http://en.wikipedia.org/wiki/Joseph_Marie_Jacquard

Jacquard loom (1805)



http://en.wikipedia.org/wiki/Jacquard_loom

The theoretical basis of imperative programming was already founded in the 30s.

Alan Turing

http://en.wikipedia.org/wiki/Alan_Turing

John von Neumann

http://en.wikipedia.org/wiki/John_von_Neumann

The theory of functions as a model for calculation comes also from the 20s and 30s.

Moisei Sheinfinkel
(Moses Schönfinkel)

http://ru.wikipedia.org/wiki/Шейнфинкель,_Моисей_Эльевич

Haskell Curry

http://www.haskell.org/haskellwiki/Haskell_Brooks_Curry

Alonzo Church

http://www.eecs.berkeley.edu/~bh/ssch9/lambda.html

Lambda calculus

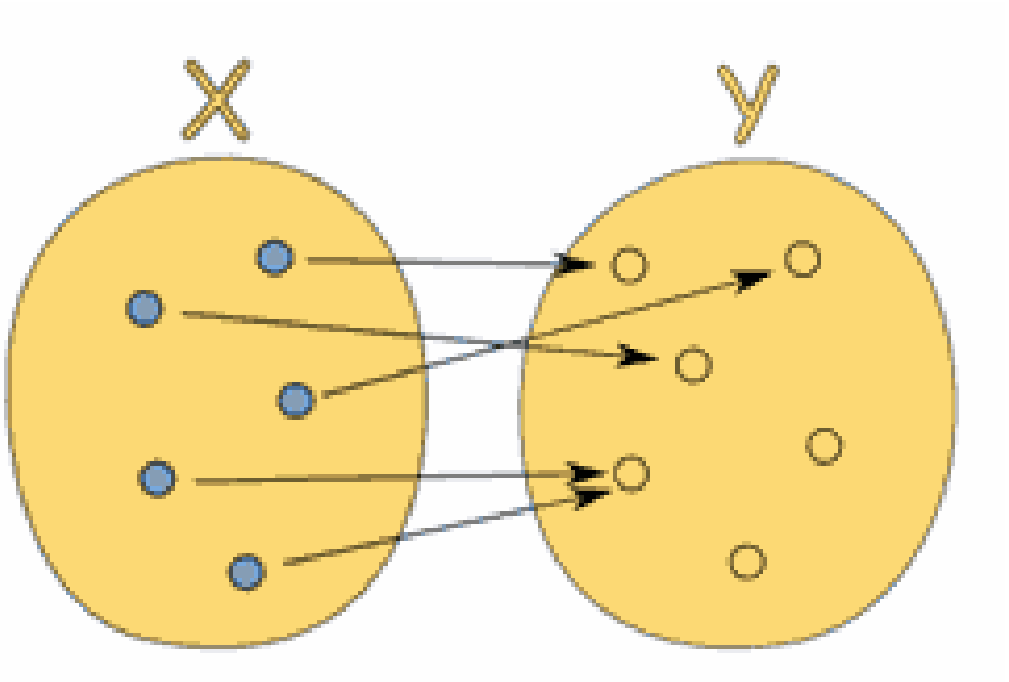
- It all began in the 1930's with Alonzo Church when he created the lambda calculus.

He wanted to discribe the world in functions.

Lambda …

http://wingolog.org/archives/2011/07/12/static-single-assignment-for-functional-programmers/

Formal definition of a function



http://www.mathsisfun.com/sets/domain-range-codomain.html

The square

square(1) = 1 * 1 = 1

square(2) = 2 * 2 = 4

…

…

…

square(x) ≡ x * x

f(x) = x * x

Lambda calculus

- square(x) ≡ x*x
- f(x) = x*x
- λ(x) = x*x

$$\lambda x.x*x$$

argument body

Church encoding

- 0 := λf.λx.x
- 1 := λf.λx.f x
- 2 := λf.λx.f (f x)
- 3 := λf.λx.f (f (f x))

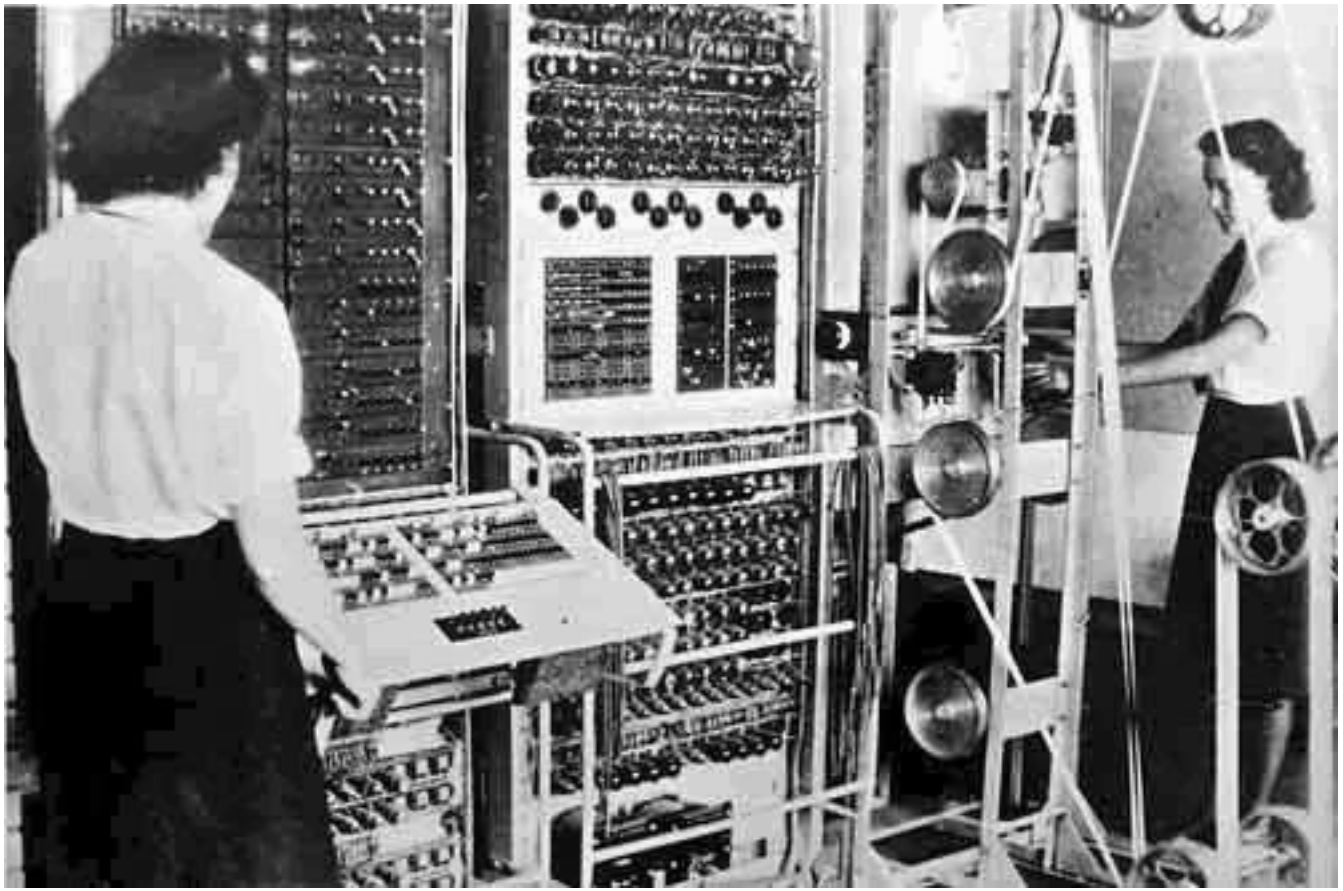
- 0 -> x
- 1 -> f(x)
- 2 -> f(f(x))
- 3 -> f(f(f(x)))
- N+1 -> f(N)

World War II

http://bigpicture.ru/?p=204724

In the 40s the first (electric programmable) computers were built.

Colossus computer



http://en.wikipedia.org/wiki/Colossus_computer

Colossus computer

It were used by British codebreakers during World War II

to help in the cryptanalysis of the Lorenz cipher.

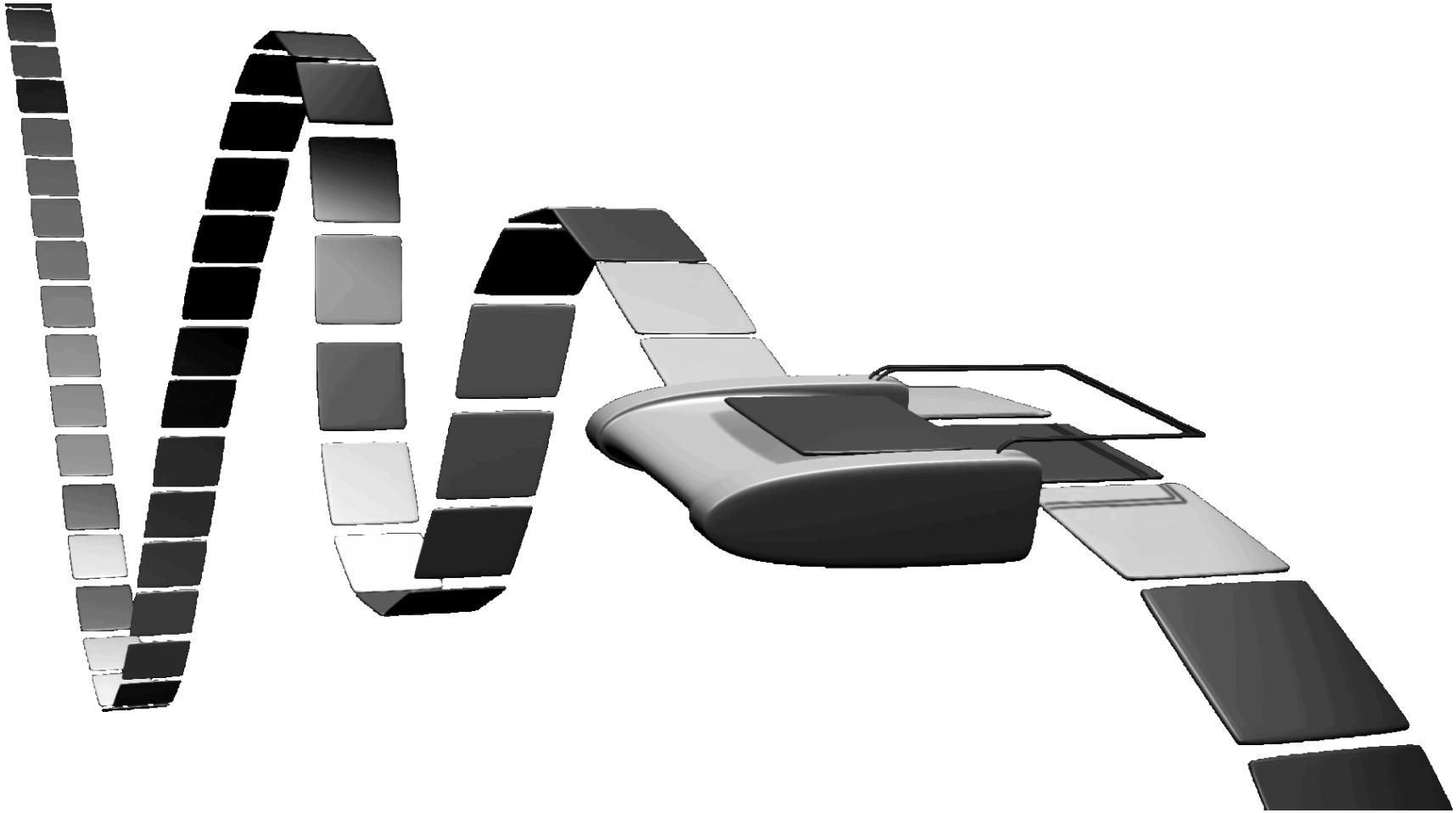
The first computers

In those days computer use was very expensive.

It was obvious to have the programming language resemble the architecture of the computer as close as possible.

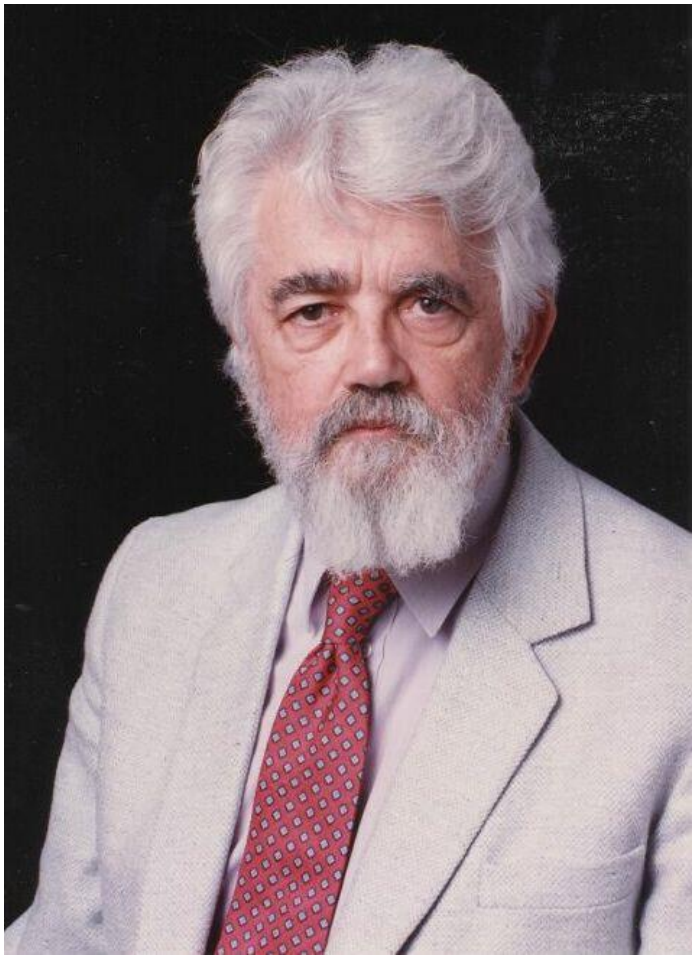
Focused only on the works of Alan Turing.

The turing machine



https://en.wikipedia.org/wiki/Turing_machine

John McCarthy



http://www-formal.stanford.edu/jmc/personal.html

LISP (1958)

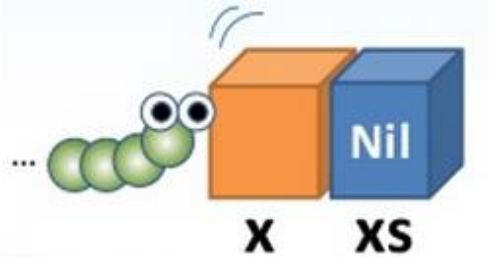
The name LISP derives from "LISt Processing".

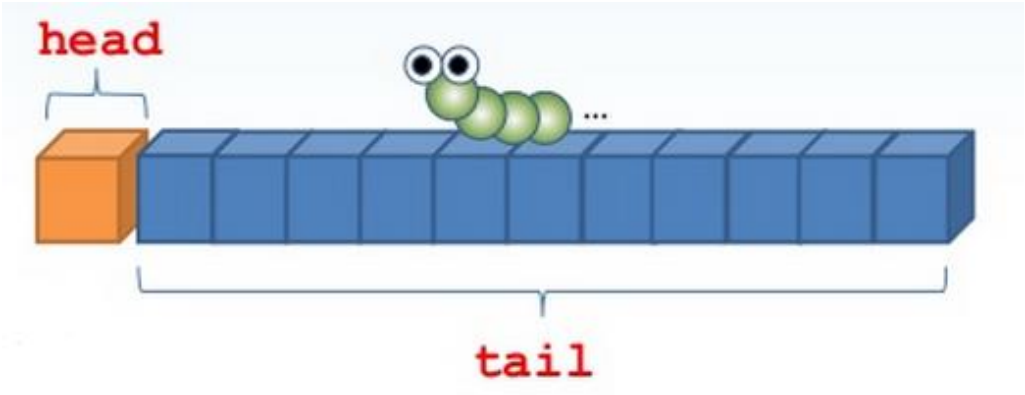
- Lots of Irritating Superfluous Parentheses
- Lost In Stupid Parentheses

A linked lists

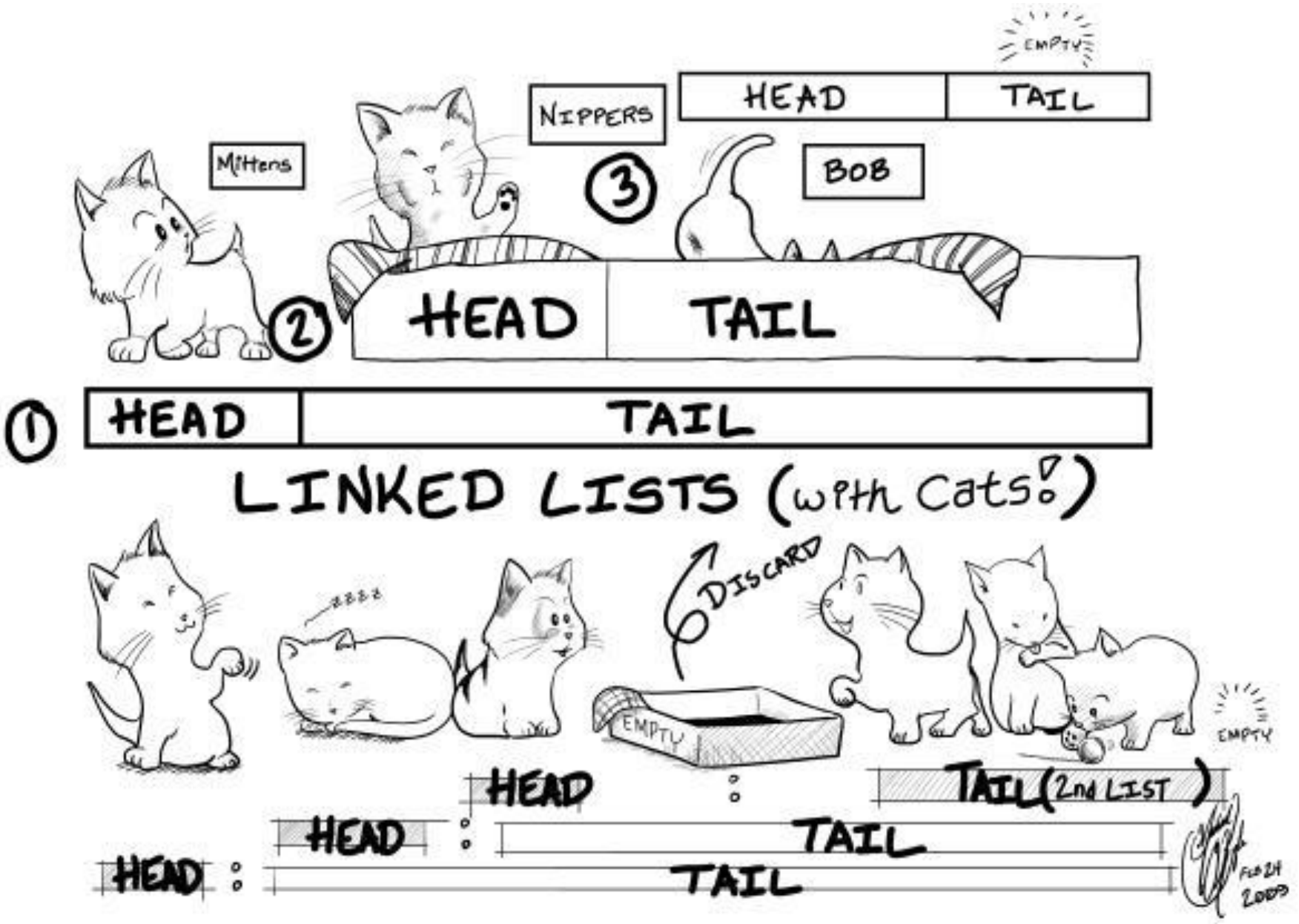
Linked lists are one of Lisp language's major data structures, and Lisp source code is itself made up of lists.

A linked lists





http://openhome.cc/eGossip/Blog/FunctionalProgrammingforJavaDevelopers2.html



http://fp.okeefecreations.com/2009/02/counting-cats-and-listing-snakes.html

Jan Łukasiewicz

http://en.wikipedia.org/wiki/Jan_Łukasiewicz

A polish notation

It’s a form of notation for logic, arithmetic, and algebra.

+ 3 4

http://en.wikipedia.org/wiki/File:Prefix-dia.svg

Symbolic expressions (S-expressions)

A code in LISP
is written
as
S-expressions.

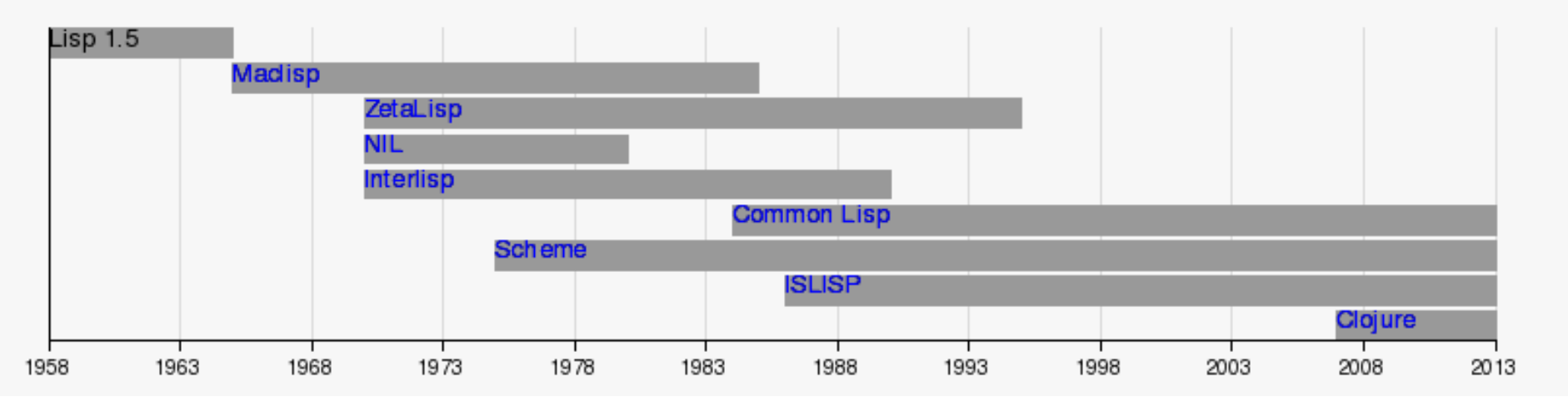
S-expressions

- (A B C)
- (VeryLongLongName)
- (A (B C))

Lisp pioneered many ideas

- tree data structures
- automatic storage management
- dynamic typing
- conditionals
- higher-order functions
- recursion
- the self-hosting compiler
- …

The LISP dialects



https://en.wikipedia.org/wiki/Lisp_(programming_language)

Peter Landin

http://www.guardian.co.uk/technology/2009/sep/22/peter-landin-obituary

Peter Landin’s work in the mid 60’s was the next significant impetus to the functional programming paradigm.

ISWIM (1966)

If you See What I Mean.

- Also said to have stood for "I See What You Mean", but ISWYM was mistyped as ISWIM.

http://www.cs.cmu.edu/~crary/819-f09/Landin66.pdf

PAL (1968)

Pedagogic Algorithmic Language

- The PAL language is a direct descendent of Peter Landin's ISWIM, although there are important differences, particularly in the imperatives.

http://www.softwarepreservation.org/projects/lang/PAL/Notes_on_Programming_Linguistics.pdf

David Turner

http://www.cs.kent.ac.uk/people/staff/dat/luke.html

SASL (1973)

The name SASL stands for "St Andrews Static Language"

- SASL is a mathematical notation for describing certain kinds of data structure.

SASL (1973)

- In SASL a "program" is an expression describing some data object.

```
2+3?
```

is a SASL program, to which the system responds by printing:

```
5
```

SASL (1973)

A slightly more complicated example will serve better to convey the flavour of the language

```
fac 4
    where fac 0 = 1
    fac n = n * fac (n – 1)
?
```

SASL (1973)

"fac 4" yields in succession the expressions:

4 * fac 3

4 * (3 * fac 2)

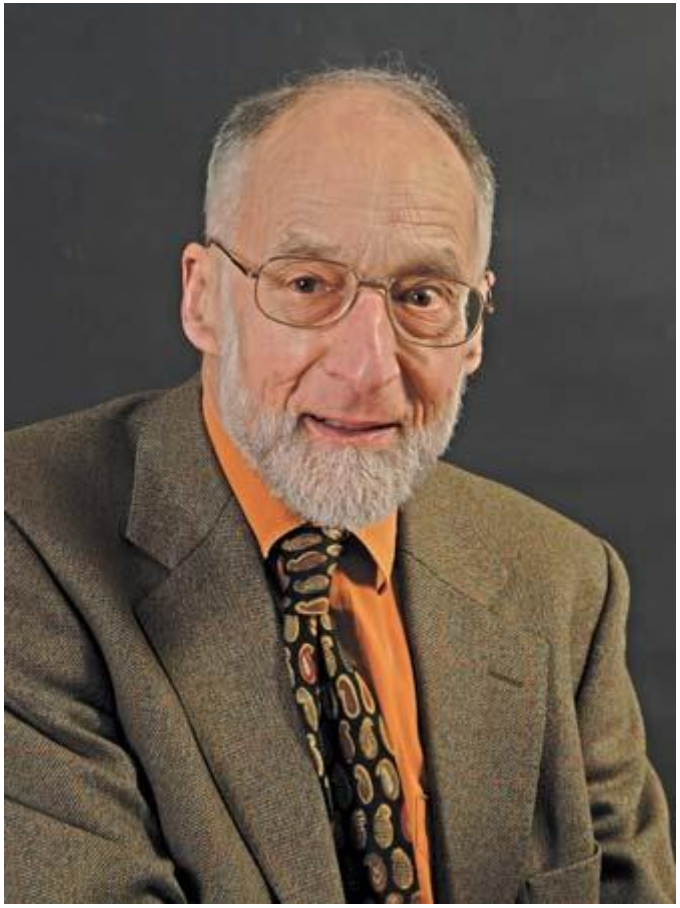
4 * (3 * (2 * fac 1))

4 * (3 * (2 * (1 * fac 0)))

4 * (3 * (2 * (1 * 1)))

which gives the value 24

Robin Milner



http://www.britannica.com/EBchecked/media/154636/Robin-Milner

ML (1973)

ML is a general-purpose functional programming language.

- Syntax is inspired by ISWIM.

ML (1973)

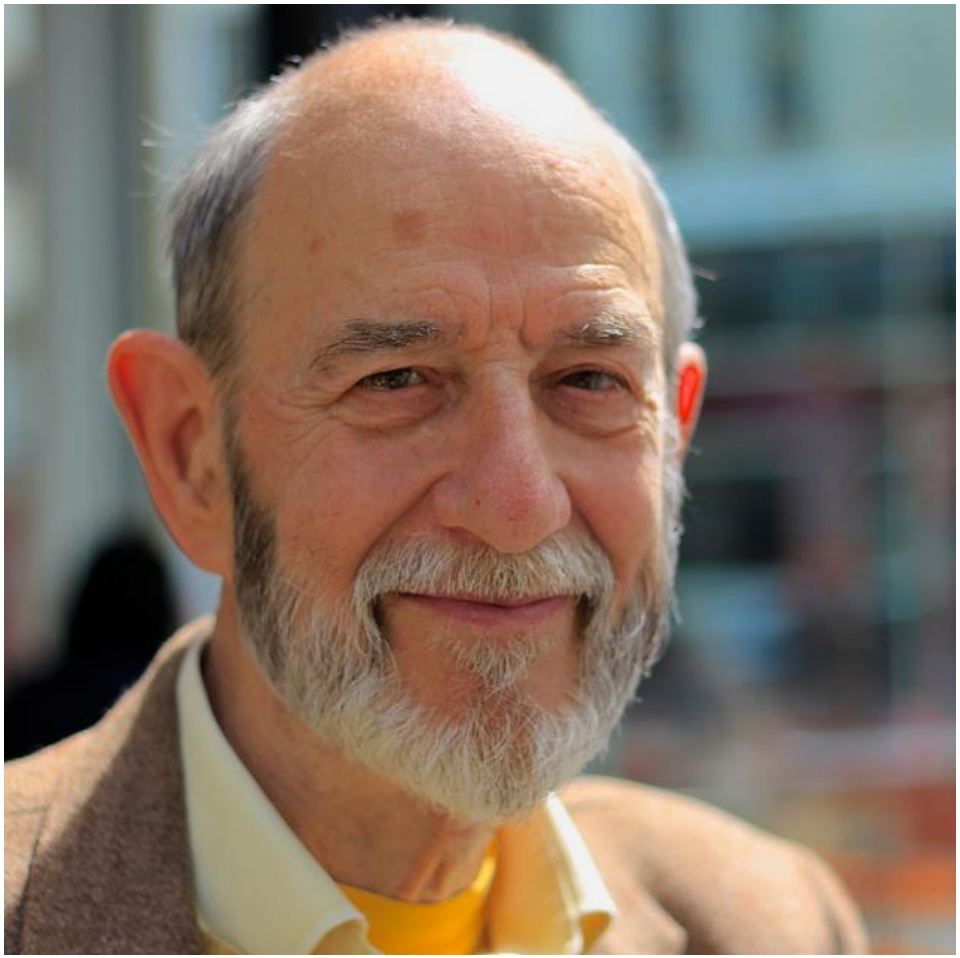
It was conceived to develop proof tactics in the LCF theorem prover.

- ML is often referred to as an impure functional language

ML influenced

- Miranda
- Haskell
- Cyclone
- C++
- F#
- Clojure
- Erlang
- …

Rod Burstall



http://www.flickr.com/photos/r_topor/7886049392/

NPL (1977)

NPL was a functional programming language with pattern matching.

http://en.wikipedia.org/wiki/NPL_programming_language

Hope (~80)

The name may have been derived from Hope Park Square in Edinburgh,

at one time the location of the Department of Artificial Intelligence.

http://en.wikipedia.org/wiki/Hope_programming_language

Hope (~80)

Language with polymorphic typing, algebraic types, pattern matching and higher-order functions.

- It predates Miranda and Haskell and is contemporaneous with ML.

http://en.wikipedia.org/wiki/Hope_programming_language

David Turner

http://www.cs.kent.ac.uk/people/staff/dat/luke.html

Miranda (1985)

Miranda is a lazy, purely functional programming language.

- The later Haskell programming language is similar in many ways to Miranda.

Each group develops its own language

http://www.thelasthonestguy.com/2011/04/how-many-guys-is-“too”-many/

Haskell (1987)

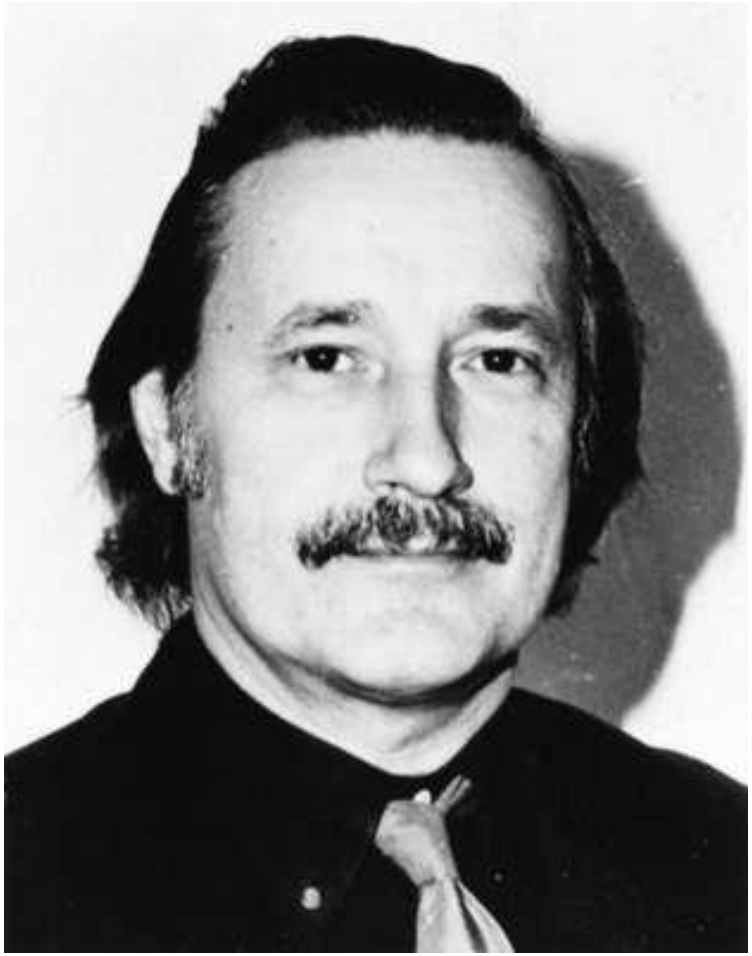
It’s a standardized, general-purpose purely functional programming language, with non-strict semantics and strong static typing.

http://en.wikipedia.org/wiki/Haskell_(programming_language)

Concepts of functional programming

- First-class and higher-order functions
- Pattern matching
- Single assignment
- Lazy evaluation
- Garbage collection
- List comprehensions
- …

Christopher Strachey



http://dada.compart-bremen.de/node/4438

First-class function

The term was coined by Christopher Strachey in the context of “functions as first-class citizens” in the mid-1960s

First-class function

Features can be stored in variables, passed as arguments to functions, created within functions and returned from functions.

First-class function

- map :: (a -> b) -> [a] -> [b]
- map f []   = []
- map f (x:xs) = f x : map f xs

Higher order function

The complete concept of higher-order functions is based two ideas:

- functions can be passed as arguments
- a normal function (or even nested functions) can be returned from a function.

Higher order function

Language		Higher-order functions		Non-local variables			Partial application	Notes
		Arguments	Results	Nested functions	Anonymous functions	Closures		
Algol family	ALGOL 60	Yes	No	Yes	No	No	No	Have function types.
	ALGOL 68	Yes	Yes[8]	Yes	Yes	No	No	
	Pascal	Yes	No	Yes	No	No	No	
	Oberon	Yes	Non-nested only	Yes	No	No	No	
C family	C	Yes	Yes	No	No	No	No	Has function pointers.
	C++	Yes	Yes	No	C++11[9]	C++11[9]	No	Has function pointers, function objects. (Also, see below.)
	C#	Yes	Yes	No	2.0 / 3.0	2.0	No	Has delegates (2.0) and lambda expressions (3.0).
	Objective-C	Yes	Yes	No	2.0 + Blocks[10]	2.0 + Blocks	No	Has function pointers.
	Java	Partial	Partial	No	No	8	No	Has anonymous inner classes.
Functional languages	Lisp	Syntax	Syntax	Yes	Yes	Common Lisp	No	(see below)
	Scheme	Yes	Yes	Yes	Yes	Yes	SRFI 26[11]	
	Clojure	Yes	Yes	Yes	Yes	Yes	Yes	
	ML	Yes	Yes	Yes	Yes	Yes	Yes	
	Haskell	Yes	Yes	Yes	Yes	Yes	Yes	
	Scala	Yes	Yes	Yes	Yes	Yes	Yes	
Scripting languages	JavaScript	Yes	Yes	Yes	Yes	Yes	ECMAScript 5	Partial application possible with user-land code on ES3 [12]
	PHP	Yes	Yes	Unscoped	5.3	5.3	No	(see below)
	Perl	Yes	Yes	anonymous, 6	Yes	Yes	6[13]	(see below)
	Python	Yes	Yes	Yes	Partial	Yes	2.5[14]	(see below)
	Ruby	Syntax	Syntax	Unscoped	Yes	Yes	1.9	(see below)
Other languages	Mathematica	Yes	Yes	Yes	Yes	Yes	No	
	Smalltalk	Yes	Yes	Yes	Yes	Yes	Partial	Partial application possible through library.
	Fortran	Yes	Yes	Yes	No	No	No	

A **lexical closures**

- It’s a function that can refer to and alter the values of bindings established by binding forms that textually include the function definition.
- It’s often referred to just as a closure

A **pattern matching**

http://magoosh.com/gmat/common-gmat-math-mistakes/

A **pattern matching**

It’s a dispatch mechanism: choosing which variant of a function is the correct one to call.

- The patterns generally have the form of either sequences or tree structures.
- Inspired by standard mathematical notations.
- Pattern matching is not a switch.

A **single assignment**

When a variable is assigned once, at most.

- It's also called *initialization*.
- It does not make sense in an **imperative programming**.

A **lazy evaluation**

A **lazy evaluation**

An evaluation should be delayed as long as they do not need the result.

- It's also known as **call-by-need**.
- It’s opposite a **strict evaluation**.

A **lazy evaluation**

- lazy evaluation frees a programmer from concerns about evaluation order
- the ability to compute with unbounded (“infinite”) data structures

A **garbage collection**

It's a form of automatic memory management.

- It attempts to reclaim memory occupied by objects that are no longer in use by the program.
- It was invented by John McCarthy around 1959 to solve problems in Lisp

A **garbage collection**

There are three main techniques for automatic memory management:

- reference counting
- mark-and-sweep
- copying.

A **list comprehensions**

It's *syntactic sugar* for a combination of applications of the functions concat, map and filter.

A **list comprehensions** in Haskell

```
pyth n = [ ( a, b, c ) | a <- [1..n],
                         b <- [1..n-a+1],
                         c <- [1..n-a-b+2],
                         a + b + c <= n,
                         a^2 + b^2 == c^2 ]
```

The key idea

Do everything by composing functions:

- no mutable state
- no side effects

In summary

This is a way to create a program in which:

- the only one action is to call a function
- the only one way partition is a function
- the only one rule of composition - the composition of functions

No memory nor assignment operators nor cycle nor block diagrams nor flow control ...

Resource

Books and papers

- **Functional Programming Application and Implementation** by *Peter Henderson*
- **A Unification of Functional and Imperative Languages** by *Ben Zhang*
- **The Conception, Evolution, and Application of Functional Programming Languages** by *Paul Hudak (Yale University)*
- **SASL LANGUAGE MANUAL** by *David Turner*
- **PAL -- A Reference Manual and a Primer** by *A. Evans*