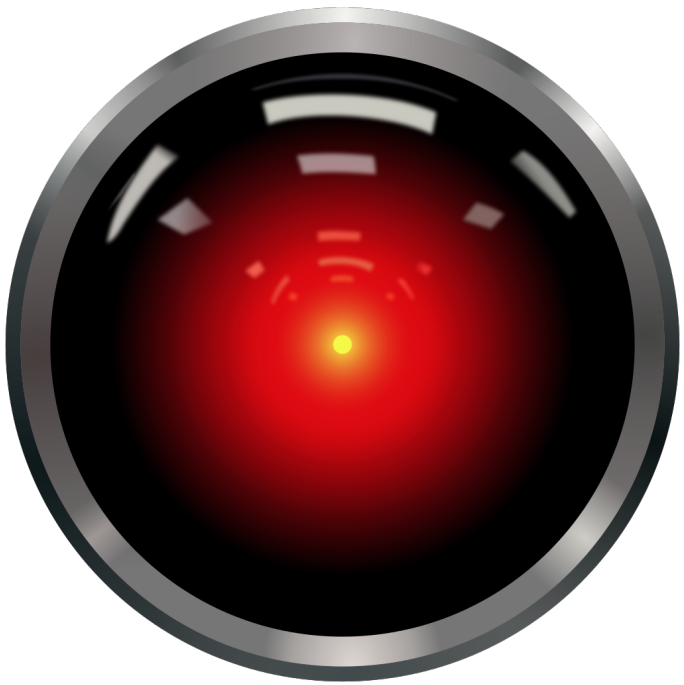
HAL

Héctor Ramón and Alvaro Espuña

4th june, 2014



Objectve

Create a programming language:

- That results useful in the future
- Easy to write and read
- Very dynamic
- Inspired in good ideas of other languages (`Python`, `Ruby`,...)

```
5.times: print "Hello world!"
```

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

Main features

- **Clean** syntax, perfect for creating **D**omain-**S**pecific **L**anguages
- **Object-oriented** with **inheritance**
- **Dynamic typing** and **duck typing**
- **Builtin** methods that can be **overriden** in `HAL` itself
- Module **imports**
- **First-class** functions
- Interactive and easy-to-extend interpreter

Clean syntax

```
class Array:
  def sort!:
    return self if size < 2
    p = first
    q = pop!
    lesser = q.filter with x: x < p
    greater = q.filter with x: x >= p
    lesser.sort! ++ [p] ++ greater.sort!

a = [1, 2, 3, -1, -2, -3, 20, 40, 1, 2, 200, -5]
print a.sort!
```

```
[-5, -3, -2, -1, 1, 1, 2, 2, 3, 20, 40, 200]
```

List comprehension

```
class Array:
  def map:
    [ yield x for x in self ]

a = range(10).map with x: x * 2
print a
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

First-class functions

```
def lambda: &yield

add = lambda with x, y: x + y

print add 5, add 3, add 1, 2
print &add.arity
```

```
11
2
```

Lambda blocks

```
class Integer:
  def times:
    i = 0
    while i < self:
      yield if &yield.arity == 0 else yield i
      i = i + 1

5.times: print "Something"
3.times with t: print "%s" % t
```

```
Something
Something
Something
Something
Something
0
1
2
```

Backticks

```
File.open "test.txt" with f:
  f.write "Writing a file has never been so easy!"

print `cat test.txt`
```

```
Writing a file has never been so easy!
```

Spaces are significative

```
def f x: x * 20
a = 10

print \
   a - 1,
   a-1,
   f -1
```

```
9
9
-20
```

Four levels of scopes

Local	Without *accessor*
Instance	Using `@`
Static	Using `@@`
Module	Instance variables in the current module

When a name is referenced without any *accessor*, `HAL` looks in the scopes in that order.

Builtin classes

- Boolean
- Class
- Enumerable
  - Array
  - Dictionary
  - String
- File
- Kernel
  - Module
- None
- Number
  - Float
  - Integer
  - Long
  - Rational
- Object
- Package
- Process

Basic data types

```
public abstract class HalNumber<T extends Number>
    extends HalObject<T> {
  public abstract HalNumber add(HalNumber n);

  public abstract boolean canCoerce(HalObject n);
  public abstract HalNumber coerce(HalObject n);
```

Basic data types (I)

```
private static final Reference __add__ =
  new Reference(new Builtin("add", new Params.Param("x")) {
    @Override
    public HalObject mcall(HalObject instance,
      HalMethod lambda, Arguments args) {
       HalNumber i = ((HalNumber) instance);
       HalObject x = args.get("x");

            if (!i.canCoerce(x))
                return x.methodcall("__radd__", i);

            return i.add(i.coerce(x));
        }
    });
```

Basic data types (II)

```
public static final HalClass klass =
    new HalClass("Number", HalObject.klass,
   // ...
   __add__,
   // ...
);
```

```
class HalInteger extends HalNumber<Integer> {
 // ...
 @Override
 public HalNumber add(HalNumber n) {
   if (addOverflows(toInteger(), n.toInteger()))
     return new HalLong(toInteger())
             .add(new HalLong(n.toInteger()));
   return new HalInteger(toInteger() + n.toInteger());
 }
 // ...
}
```

Basic data types (III)

Other `Number` methods:

- `bool`
- `pos`
- `neg`
- `sub`
- `mul`
- `pow`
- `div`
- `mod`
- `ddiv`
- `eq`
- `lt`

HALTeX

```
import haltex

section 'Use example'
enumerate:
  item; p '*First item*'
  item; p '|second|'
  item; p '**third**'
```

```
\section{Use example}
\begin{enumerate}
\item
\emph{First item}
\item
\texttt{second}
\item
\textbf{third}
\end{enumerate}
```