

Lisp

Syntax

So, is `(+ 1 2)` a list of three things or code to add two numbers?

Lisp

Syntax

So, is `(+ 1 2)` a list of three things or code to add two numbers?

Both!

Lisp

Program and data are identical in Lisp

Lisp

Syntax

This makes Lisp a particularly powerful language

Lisp

Syntax

This makes Lisp a particularly powerful language

Lisp programs can trivially manipulate other Lisp programs

Lisp

Syntax

This makes Lisp a particularly powerful language

Lisp programs can trivially manipulate other Lisp programs

...or even themselves

Lisp

Syntax

This makes Lisp a particularly powerful language

Lisp programs can trivially manipulate other Lisp programs

... or even themselves

Lisp compilers and interpreters are usually written in Lisp

Lisp

Syntax

This makes Lisp a particularly powerful language

Lisp programs can trivially manipulate other Lisp programs

... or even themselves

Lisp compilers and interpreters are usually written in Lisp

In fact, in many Lisps there is a function called `eval` that takes some Lisp code (a list) and evaluates it

Lisp

Syntax

When you use a Lisp interpreter it is essentially running this:

```
(print (eval (read)))
```

in a loop

Lisp

Syntax

When you use a Lisp interpreter it is essentially running this:

```
(print (eval (read)))
```

in a loop

Namely, read an expression, evaluate it, then print the result:
often called a REP loop

Lisp

Syntax

When you use a Lisp interpreter it is essentially running this:

```
(print (eval (read)))
```

in a loop

Namely, read an expression, evaluate it, then print the result:
often called a REP loop

Some Lisps do not allow user programs to run `eval` as there are some interesting issues that surround the function

Lisp

Syntax

When you use a Lisp interpreter it is essentially running this:

```
(print (eval (read)))
```

in a loop

Namely, read an expression, evaluate it, then print the result:
often called a REP loop

Some Lisps do not allow user programs to run `eval` as there are some interesting issues that surround the function

Not least you can change or provide an alternative definition of `eval`

Lisp

Syntax

Think about this: Lisp is a language that allows you to change the way it works

Lisp

Syntax

Think about this: Lisp is a language that allows you to change the way it works

As it runs

Lisp

Syntax

Think about this: Lisp is a language that allows you to change the way it works

As it runs

But don't! Most people have problems writing programs when they think they understand what an expression means: if that changes underfoot you have no chance

Lisp

Syntax

You could even redefine `read` to allow a different syntax to Lisp:
see Rlisp

Lisp

Syntax

You could even redefine `read` to allow a different syntax to Lisp:
see Rlisp

Some languages, e.g., ML and Lua, are fundamentally Lisp
with an Algol syntax

Lisp

Syntax

You could even redefine `read` to allow a different syntax to Lisp:
see `Rlisp`

Some languages, e.g., ML and Lua, are fundamentally Lisp
with an Algol syntax

The fact that most people don't change Lisp is because the
parenthesis syntax is actually quite useful

Lisp

Syntax

You could even redefine `read` to allow a different syntax to Lisp:
see `Rlisp`

Some languages, e.g., ML and Lua, are fundamentally Lisp
with an Algol syntax

The fact that most people don't change Lisp is because the
parenthesis syntax is actually quite useful

People *do* change `eval` to allow, say, the introduction of an OO
system

Lisp

Syntax

You could even redefine `read` to allow a different syntax to Lisp:
see Rlisp

Some languages, e.g., ML and Lua, are fundamentally Lisp
with an Algol syntax

The fact that most people don't change Lisp is because the
parenthesis syntax is actually quite useful

People *do* change `eval` to allow, say, the introduction of an OO
system

Many ideas are first tried out in Lisp before being moved into a
newly designed language

Lisp

Syntax

So `(+ 1 2)` is a list of three objects, that when given to the function `eval` it returns the value 3

Lisp

Syntax

So `(+ 1 2)` is a list of three objects, that when given to the function `eval` it returns the value 3

It's a matter of context: if you ask `eval` to evaluate it, it's code; else it's a list

Lisp

Syntax

Another consequence of the malleability of Lisp is that everybody goes and makes their own version

Lisp

Syntax

Another consequence of the malleability of Lisp is that everybody goes and makes their own version

There are a large number of languages out there that could be called “Lisp”

Lisp

Syntax

Another consequence of the malleability of Lisp is that everybody goes and makes their own version

There are a large number of languages out there that could be called “Lisp”

Generally “Lisp” is thought of a family, rather than a single thing

Lisp

Syntax

Another consequence of the malleability of Lisp is that everybody goes and makes their own version

There are a large number of languages out there that could be called “Lisp”

Generally “Lisp” is thought of a family, rather than a single thing

With C and Java you know pretty well what you are getting: there are standards definitions that implementations of these languages are supposed to follow

Lisp

Syntax

Another consequence of the malleability of Lisp is that everybody goes and makes their own version

There are a large number of languages out there that could be called “Lisp”

Generally “Lisp” is thought of a family, rather than a single thing

With C and Java you know pretty well what you are getting: there are standards definitions that implementations of these languages are supposed to follow

With Lisp there's all kinds of variation

Lisp

Diaspora

Each Lisp does everything in its own special way

Lisp

Diaspora

Each Lisp does everything in its own special way

In Cambridge Lisp the function to add numbers is called `plus`;
in Common Lisp it is called `+`

`(plus 1 2)` vs. `(+ 1 2)`

Lisp

Diaspora

Each Lisp does everything in its own special way

In Cambridge Lisp the function to add numbers is called `plus`;
in Common Lisp it is called `+`

`(plus 1 2)` vs. `(+ 1 2)`

These are superficial differences

Lisp

Diaspora

The “if” construct might have an optional “else” part:

```
(if (> x 1) (print "hello"))
```


Lisp

Diaspora

The “if” construct might have an optional “else” part:

```
(if (> x 1) (print "hello"))
```

Or it might require it

```
(if (> x 1) (print "hello") (print "bye"))
```

and provide an alternative, single clause “if”

```
(when (> x 1) (print "hello"))
```

Lisp

Diaspora

The “if” construct might have an optional “else” part:

```
(if (> x 1) (print "hello"))
```

Or it might require it

```
(if (> x 1) (print "hello") (print "bye"))
```

and provide an alternative, single clause “if”

```
(when (> x 1) (print "hello"))
```

And so on

Lisp

Diaspora

The *semantics* of everything is roughly the same, so generally things do what you expect of them

Lisp

Diaspora

The *semantics* of everything is roughly the same, so generally things do what you expect of them

Though they don't have to...

Lisp

Diaspora

The *semantics* of everything is roughly the same, so generally things do what you expect of them

Though they don't have to. . .

This makes portability of programs an issue, but has helped immensely in the development of new ideas

Lisp

Diaspora

The *semantics* of everything is roughly the same, so generally things do what you expect of them

Though they don't have to. . .

This makes portability of programs an issue, but has helped immensely in the development of new ideas

Lisp has been called a “ball of mud”, meaning you can throw anything at it—and you just get a larger ball of mud

Lisp

Diaspora

The *semantics* of everything is roughly the same, so generally things do what you expect of them

Though they don't have to. . .

This makes portability of programs an issue, but has helped immensely in the development of new ideas

Lisp has been called a “ball of mud”, meaning you can throw anything at it—and you just get a larger ball of mud

Lisps come in all kinds of shapes and sizes: but they are all Lisps

Lisp

Standards

There are in fact more than a few standards for Lisp

Lisp

Standards

There are in fact more than a few standards for Lisp

The two widely used ones are

- Common Lisp
- Scheme

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

It arose when the US defence research agency ARPA wanted a single Lisp it could use

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

It arose when the US defence research agency ARPA wanted a single Lisp it could use

At the time there were many Lisps floating about and ARPA wanted a single standard it could write programs for

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

It arose when the US defence research agency ARPA wanted a single Lisp it could use

At the time there were many Lisps floating about and ARPA wanted a single standard it could write programs for

Many Lisp implementors and vendors were called together to create a standard

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

It arose when the US defence research agency ARPA wanted a single Lisp it could use

At the time there were many Lisps floating about and ARPA wanted a single standard it could write programs for

Many Lisp implementors and vendors were called together to create a standard

After a huge amount of wrangling, Common Lisp emerged

Lisp

Standards

Common Lisp is a large standard describing a huge Lisp

It arose when the US defence research agency ARPA wanted a single Lisp it could use

At the time there were many Lisps floating about and ARPA wanted a single standard it could write programs for

Many Lisp implementors and vendors were called together to create a standard

After a huge amount of wrangling, Common Lisp emerged

Roughly speaking, Common Lisp is the union of all the features of all the Lisps: no vendor wanted their special features to be left out

Lisp

Standards

So, for example, there are two functions to remove an element from a list: `delete` and `remove`

Lisp

Standards

So, for example, there are two functions to remove an element from a list: `delete` and `remove`

They do different things to the list: one updates the list to remove the element; the other creates a new list that is a copy without the element

Lisp

Standards

So, for example, there are two functions to remove an element from a list: `delete` and `remove`

They do different things to the list: one updates the list to remove the element; the other creates a new list that is a copy without the element

So Common Lisp provides a rich array of functionality

Lisp

Standards

So, for example, there are two functions to remove an element from a list: `delete` and `remove`

They do different things to the list: one updates the list to remove the element; the other creates a new list that is a copy without the element

So Common Lisp provides a rich array of functionality

This was version 1, as documented in the book “Common Lisp: The Language” (CLtL1)

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

This became an ANSI standard: X3.226-1994 (R1999)

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

This became an ANSI standard: X3.226-1994 (R1999)

This is reasonably decent as a standard, but is huge at over 1000 pages

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

This became an ANSI standard: X3.226-1994 (R1999)

This is reasonably decent as a standard, but is huge at over 1000 pages

A large chunk of this a list of functions and their required behaviours (like `delete` and `remove`)

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

This became an ANSI standard: X3.226-1994 (R1999)

This is reasonably decent as a standard, but is huge at over 1000 pages

A large chunk of this a list of functions and their required behaviours (like `delete` and `remove`)

But there is important stuff in there, too, such as the specification of the behaviour of functions over the Complex numbers

Lisp

Standards

After more work, version 2 emerged, “Common Lisp: The Language, Second Edition” (CLtL2)

This became an ANSI standard: X3.226-1994 (R1999)

This is reasonably decent as a standard, but is huge at over 1000 pages

A large chunk of this a list of functions and their required behaviours (like `delete` and `remove`)

But there is important stuff in there, too, such as the specification of the behaviour of functions over the Complex numbers

The Java standard is now larger. . .

Lisp

Standards

Meanwhile other people (mostly academics) were saying: this is too big, what we need is simplicity

Lisp

Standards

Meanwhile other people (mostly academics) were saying: this is too big, what we need is simplicity

They defined *Scheme* in a document called “The Report on Scheme”

Lisp

Standards

Meanwhile other people (mostly academics) were saying: this is too big, what we need is simplicity

They defined *Scheme* in a document called “The Report on Scheme”

Roughly, this was the intersection of all current Lisps

Lisp

Standards

Meanwhile other people (mostly academics) were saying: this is too big, what we need is simplicity

They defined *Scheme* in a document called “The Report on Scheme”

Roughly, this was the intersection of all current Lisps

Schemers claim Scheme is not Lisp, but it is certainly of the family

Lisp

Standards

Meanwhile other people (mostly academics) were saying: this is too big, what we need is simplicity

They defined *Scheme* in a document called “The Report on Scheme”

Roughly, this was the intersection of all current Lisps

Schemers claim Scheme is not Lisp, but it is certainly of the family

To be included in Scheme, a feature must be essential and not implementable in terms of existing features

Lisp

Standards

The Scheme standard was revised: called “The Revised Report on Scheme”

Lisp

Standards

The Scheme standard was revised: called “The Revised Report on Scheme”

Then to “The Revised Revised Report on Scheme”, or R2RS

Lisp

Standards

The Scheme standard was revised: called “The Revised Report on Scheme”

Then to “The Revised Revised Report on Scheme”, or R2RS

And so on

Lisp

Standards

The Scheme standard was revised: called “The Revised Report on Scheme”

Then to “The Revised Revised Report on Scheme”, or R2RS

And so on

R5RS is just 50 pages long

Lisp

Standards

The Scheme standard was revised: called “The Revised Report on Scheme”

Then to “The Revised Revised Report on Scheme”, or R2RS

And so on

R5RS is just 50 pages long

R6RS includes (not to universal acclaim) specifications of library functions, so is longer, but the basic language part is just 90 pages long

Lisp

Standards

R7RS (2013) has split the language into chunks, “large” and “small”

Lisp

Standards

R7RS (2013) has split the language into chunks, “large” and “small”

The small part much closer to R5RS (88 pages)

Lisp

Standards

R7RS (2013) has split the language into chunks, “large” and “small”

The small part much closer to R5RS (88 pages)

The large is “focused on the practical needs of mainstream software development”, and is closer to R6RS

Lisp

Standards

Scheme is characterised by having few, but powerful, constructs

Lisp

Standards

Scheme is characterised by having few, but powerful, constructs

For example, *continuations*

Lisp

Standards

Scheme is characterised by having few, but powerful, constructs

For example, *continuations*

A continuation is a generalisation of the idea of “current execution position in the program”

Lisp

Standards

Scheme is characterised by having few, but powerful, constructs

For example, *continuations*

A continuation is a generalisation of the idea of “current execution position in the program”

In Scheme, a continuation is a first class object, meaning a program can manipulate its own flow of control programmatically

Lisp

Standards

Continuations can be used to implement other, more understandable, things, like non-local jumps and parallel execution

Lisp

Standards

Continuations can be used to implement other, more understandable, things, like non-local jumps and parallel execution

But the idea is that continuations replace a collection of other concepts; and allow the implementation of new and different concepts

Lisp

Standards

Scheme is simple enough to be used as an introductory language in some University courses

Lisp

Standards

Scheme is simple enough to be used as an introductory language in some University courses

It is also sophisticated enough to be used to explain some very difficult topics

Lisp

Standards

Scheme is simple enough to be used as an introductory language in some University courses

It is also sophisticated enough to be used to explain some very difficult topics

The book “Structure and Interpretation of Computer Programs” by Abelson and Sussman should be read by all Computer Scientists

Lisp

Standards

At this point is it worthwhile making clear the difference between implementations and standards, as they are often confused for one another

Lisp

Standards

At this point is it worthwhile making clear the difference between implementations and standards, as they are often confused for one another

A standard is a document that describes how an implementation should behave

Lisp

Standards

At this point is it worthwhile making clear the difference between implementations and standards, as they are often confused for one another

A standard is a document that describes how an implementation should behave

An implementation is a program, usually a compiler or an interpreter

Lisp

Standards

At this point is it worthwhile making clear the difference between implementations and standards, as they are often confused for one another

A standard is a document that describes how an implementation should behave

An implementation is a program, usually a compiler or an interpreter

There can be several, differing, implementations of a standard

Lisp

Standards

At this point is it worthwhile making clear the difference between implementations and standards, as they are often confused for one another

A standard is a document that describes how an implementation should behave

An implementation is a program, usually a compiler or an interpreter

There can be several, differing, implementations of a standard

Just as there are many C compilers and a few Java compilers, there are many different implementations of Common Lisp and Scheme

Lisp

Standards

A program written to run in one implementation of, say, Common Lisp, *ought* to run on all other implementations of Common Lisp

Lisp

Standards

A program written to run in one implementation of, say, Common Lisp, *ought* to run on all other implementations of Common Lisp

Reality is never so neat

Lisp

Standards

There might be

Lisp

Standards

There might be

- bugs in the implementation

Lisp

Standards

There might be

- bugs in the implementation
- bugs in the standard

Lisp

Standards

There might be

- bugs in the implementation
- bugs in the standard
- things not defined or not clear in the standard

Lisp

Standards

There might be

- bugs in the implementation
- bugs in the standard
- things not defined or not clear in the standard
- things deliberately left undefined in the standard (e.g., size of an `int` in C. A portable program will not make the assumption that an `int` is 4 bytes)

Lisp

Standards

There might be

- bugs in the implementation
- bugs in the standard
- things not defined or not clear in the standard
- things deliberately left undefined in the standard (e.g., size of an `int` in C. A portable program will not make the assumption that an `int` is 4 bytes)
- deliberate features in the implementation designed for lock-in by the vendor

Lisp

Standards

All meaning that you have to be careful when porting a program to a new implementation

Lisp

Standards

All meaning that you have to be careful when porting a program to a new implementation

And this applies to *all* languages, particularly C

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

In fact, EuLisp comes in three sizes

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

In fact, EuLisp comes in three sizes

1. Small, Scheme sized

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

In fact, EuLisp comes in three sizes

1. Small, Scheme sized
2. Medium

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

In fact, EuLisp comes in three sizes

1. Small, Scheme sized
2. Medium
3. Large, but not as big as Common Lisp

Lisp

Standards

If Common Lisp is too large, Scheme is too small

Many day-to-day useful things are deliberately left out of Scheme, though R6 and R7 try to address this issue

In response, there is a middle-sized Lisp, *EuLisp*

In fact, EuLisp comes in three sizes

1. Small, Scheme sized
2. Medium
3. Large, but not as big as Common Lisp

Each Level is a subset of the next, so we can pick the size we need

Lisp

Standards

It is called “Eu”Lisp as its design was sponsored by the EU

Lisp

Standards

It is called “Eu”Lisp as its design was sponsored by the EU

We at Bath were strongly involved in this standard

Lisp

Standards

It is called “Eu”Lisp as its design was sponsored by the EU

We at Bath were strongly involved in this standard

The EU eventually lost interest, so the standard has not gained popularity

Lisp

Standards

It is called “Eu”Lisp as its design was sponsored by the EU

We at Bath were strongly involved in this standard

The EU eventually lost interest, so the standard has not gained popularity

The standard lives in a modified form as ISLisp, ISO standard ISO/IEC 13816:1997(E)

Lisp

Standards

EuLisp strongly influenced a language called Dylan, developed by Apple

Lisp

Standards

EuLisp strongly influenced a language called Dylan, developed by Apple

Dylan was eventually dropped by Apple, but provided impetus to the development of a new language, called Java

Lisp

Standards

You will also come across something called *Standard Lisp*

Lisp

Standards

You will also come across something called *Standard Lisp*

This was a standard defined with the intention that applications sticking to this standard (e.g., Algebra Systems) could be ported easily between Lisps

Lisp

Standards

You will also come across something called *Standard Lisp*

This was a standard defined with the intention that applications sticking to this standard (e.g., Algebra Systems) could be ported easily between Lisps

This left contentious elements *undefined*, e.g., some specific properties of the empty list

Lisp

Standards

You will also come across something called *Standard Lisp*

This was a standard defined with the intention that applications sticking to this standard (e.g., Algebra Systems) could be ported easily between Lisps

This left contentious elements *undefined*, e.g., some specific properties of the empty list

This meant programmers couldn't rely on having such properties and so had to avoid using them

Lisp

Standards

You will also come across something called *Standard Lisp*

This was a standard defined with the intention that applications sticking to this standard (e.g., Algebra Systems) could be ported easily between Lisps

This left contentious elements *undefined*, e.g., some specific properties of the empty list

This meant programmers couldn't rely on having such properties and so had to avoid using them

Thus making their programs more portable

Lisp

Standards

You will also come across something called *Standard Lisp*

This was a standard defined with the intention that applications sticking to this standard (e.g., Algebra Systems) could be ported easily between Lisps

This left contentious elements *undefined*, e.g., some specific properties of the empty list

This meant programmers couldn't rely on having such properties and so had to avoid using them

Thus making their programs more portable

Standard Lisp was eventually eclipsed by Common Lisp