Object Oriented Languages

Metaobject Protocols

With all these variants of OO, why should we be content with just one kind of OO within a language?

Object Oriented Languages

Metaobject Protocols

With all these variants of OO, why should we be content with just one kind of OO within a language?

Just because a language designer has said this language should have that kind of OO, should we be stuck with it?

Object Oriented Languages

Metaobject Protocols

With all these variants of OO, why should we be content with just one kind of OO within a language?

Just because a language designer has said this language should have that kind of OO, should we be stuck with it?

A *metaobject protocol* (MOP) is a means by which we describe what kind of object protocol we want

Object Oriented Languages

Metaobject Protocols

The metaobject protocol exposes the internal mechanisms of how objects are structured, how objects are initialised, how methods are chosen, how properties are inherited, whether we have inheritance, and so on

Object Oriented Languages

Metaobject Protocols

The metaobject protocol exposes the internal mechanisms of how objects are structured, how objects are initialised, how methods are chosen, how properties are inherited, whether we have inheritance, and so on

Early experiments with MOPs in Simula were developed in Smalltalk and led to CLOS: the *Common Lisp Object System*, a fully reflective object system

Object Oriented Languages

Metaobject Protocols

The metaobject protocol exposes the internal mechanisms of how objects are structured, how objects are initialised, how methods are chosen, how properties are inherited, whether we have inheritance, and so on

Early experiments with MOPs in Simula were developed in Smalltalk and led to CLOS: the *Common Lisp Object System*, a fully reflective object system

Recall: *reflective* means a system can look at itself and maybe even change itself

Object Oriented Languages

Metaobject Protocols

The metaobject protocol exposes the internal mechanisms of how objects are structured, how objects are initialised, how methods are chosen, how properties are inherited, whether we have inheritance, and so on

Early experiments with MOPs in Simula were developed in Smalltalk and led to CLOS: the *Common Lisp Object System*, a fully reflective object system

Recall: *reflective* means a system can look at itself and maybe even change itself

Exercise Look at `type()`, `dir()`, `getattr()`, etc., in Python

Object Oriented Languages

Metaobject Protocols

And the best way to describe an OO system?

Object Oriented Languages

Metaobject Protocols

And the best way to describe an OO system?

Using itself!

Object Oriented Languages

Metaobject Protocols

And the best way to describe an OO system?

Using itself!

In CLOS (as in other MOP languages) there are

- classes that describe the structure and behaviour of classes
- methods that describe how objects should be created and initialised
- methods that describe how methods are looked up
- methods that describe how methods should be inherited or overridden or combined
- and so on for all aspects of an OO system

Object Oriented Languages

Metaobject Protocols

Exercise Think about the bootstrap problem of a MOP

Object Oriented Languages

Metaobject Protocols

We shall take examples from Telos, the EuLisp Object System, as it is much simpler than CLOS

Object Oriented Languages

Metaobject Protocols

We shall take examples from Telos, the EuLisp Object System, as it is much simpler than CLOS

There is a bunch of predefined classes and methods that describe standard structure, standard inheritance, standard method selection and so on

Object Oriented Languages

Metaobject Protocols

We shall take examples from Telos, the EuLisp Object System, as it is much simpler than CLOS

There is a bunch of predefined classes and methods that describe standard structure, standard inheritance, standard method selection and so on

These implement the normal OO behaviour as you might expect from other languages

Object Oriented Languages

Metaobject Protocols

We shall take examples from Telos, the EuLisp Object System, as it is much simpler than CLOS

There is a bunch of predefined classes and methods that describe standard structure, standard inheritance, standard method selection and so on

These implement the normal OO behaviour as you might expect from other languages

In Telos, the class `<simple-class>` and its methods describe these standard things

Object Oriented Languages

Metaobject Protocols

We shall take examples from Telos, the EuLisp Object System, as it is much simpler than CLOS

There is a bunch of predefined classes and methods that describe standard structure, standard inheritance, standard method selection and so on

These implement the normal OO behaviour as you might expect from other languages

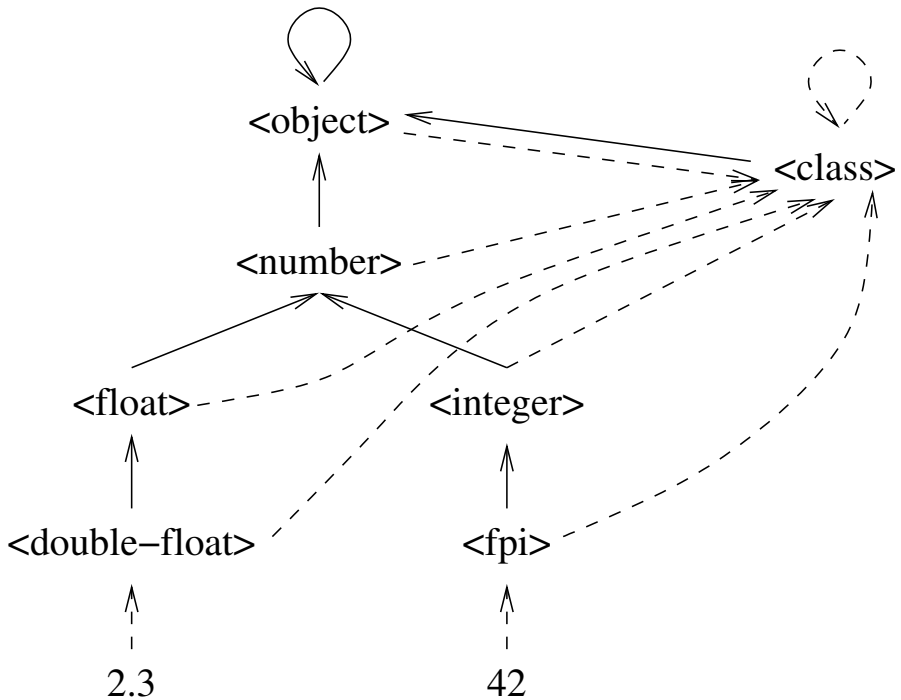
In Telos, the class `<simple-class>` and its methods describe these standard things

It is a subclass of the topmost (abstract) class `<class>`

Object Oriented Languages

Metaobject Protocols

Previously we saw:

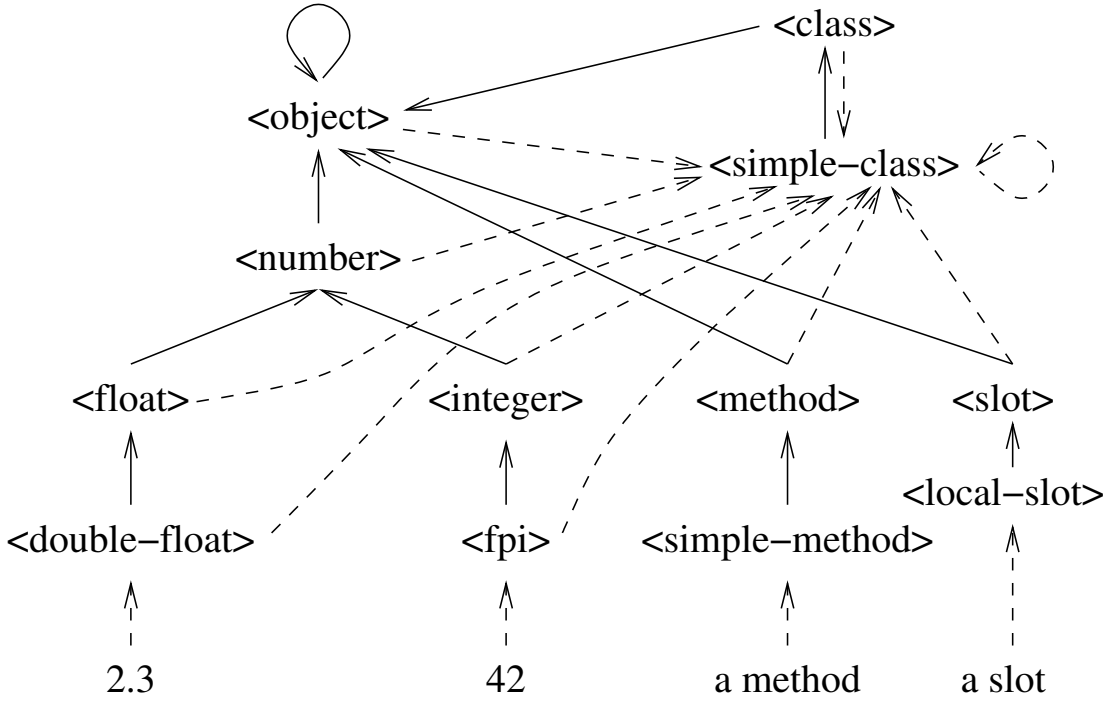


A Small Part of the EuLisp Class Hierarchy (simplified)

Dotted arrow is *instance of*/*member of*/*is a*; solid arrow is *inherits from*/*subclass*/*extends*/*subset*

Object Oriented Languages

Metaobject Protocols



A Small Part of the EuLisp Class Hierarchy (not so simplified)

Dotted arrow is *instance of* or *member of* or *is a*; solid arrow is *inherits from* or *subclass* or *extends*

Object Oriented Languages

Metaobject Protocols

If we want different behaviours we can create new classes that implement those behaviours

Object Oriented Languages

Metaobject Protocols

If we want different behaviours we can create new classes that implement those behaviours

Classes are instances of *subclasses* of the class `<class>`

Object Oriented Languages

Metaobject Protocols

If we want different behaviours we can create new classes that implement those behaviours

Classes are instances of *subclasses* of the class `<class>`

Thus the class `<string>` is an instance of `<simple-class>`

Object Oriented Languages

Metaobject Protocols

If we want different behaviours we can create new classes that implement those behaviours

Classes are instances of *subclasses* of the class `<class>`

Thus the class `<string>` is an instance of `<simple-class>`

And so strings and their methods have the “usual” behaviours, inherited from `<simple-class>`

Object Oriented Languages

Metaobject Protocols

If we want different behaviours we can create new classes that implement those behaviours

Classes are instances of *subclasses* of the class `<class>`

Thus the class `<string>` is an instance of `<simple-class>`

And so strings and their methods have the “usual” behaviours, inherited from `<simple-class>`

To define new behaviours we can create a new subclass of the class `<class>` (or `<simple-class>`)

Object Oriented Languages

Metaobject Protocols

Methods: how do we find the right method to apply?

Object Oriented Languages

Metaobject Protocols

Methods: how do we find the right method to apply?

We need to find the class precedence list for an argument

Object Oriented Languages

Metaobject Protocols

Methods: how do we find the right method to apply?

We need to find the class precedence list for an argument

So Telos provides a generic function
`compute-class-precedence-list`

Object Oriented Languages

Metaobject Protocols

Methods: how do we find the right method to apply?

We need to find the class precedence list for an argument

So Telos provides a generic function
`compute-class-precedence-list`

There is a predefined method on this for `<simple-class>` that does the standard thing with CPLs, as described previously

Object Oriented Languages

Metaobject Protocols

Methods: how do we find the right method to apply?

We need to find the class precedence list for an argument

So Telos provides a generic function
`compute-class-precedence-list`

There is a predefined method on this for `<simple-class>` that does the standard thing with CPLs, as described previously

You can add a method yourself for your new class if you want to something different, e.g., reverse the order, or omit some classes, or add some strange kind of multiple inheritance, etc.

Object Oriented Languages

Metaobject Protocols

We need to take the CPL and choose a method (or methods) from it

Object Oriented Languages

Metaobject Protocols

We need to take the CPL and choose a method (or methods) from it

The generic function `compute-method-lookup-function` is used for this

Object Oriented Languages

Metaobject Protocols

We need to take the CPL and choose a method (or methods) from it

The generic function `compute-method-lookup-function` is used for this

The standard method returns a function that simply picks the first on the list

Object Oriented Languages

Metaobject Protocols

We need to take the CPL and choose a method (or methods) from it

The generic function `compute-method-lookup-function` is used for this

The standard method returns a function that simply picks the first on the list

Method combination can be implemented by specialising `compute-method-lookup-function`

Object Oriented Languages

Metaobject Protocols

We need to take the CPL and choose a method (or methods) from it

The generic function `compute-method-lookup-function` is used for this

The standard method returns a function that simply picks the first on the list

Method combination can be implemented by specialising `compute-method-lookup-function`

And so on

Object Oriented Languages

Metaobject Protocols

Structure: how are values to be stored in an object?

Object Oriented Languages

Metaobject Protocols

Structure: how are values to be stored in an object?

Most objects are implemented as structures with the object accessors implemented as simple structure references

Object Oriented Languages

Metaobject Protocols

Structure: how are values to be stored in an object?

Most objects are implemented as structures with the object accessors implemented as simple structure references

The generic function `compute-and-ensure-slot-accessors` describes how slots are accessed

Object Oriented Languages

Metaobject Protocols

Structure: how are values to be stored in an object?

Most objects are implemented as structures with the object accessors implemented as simple structure references

The generic function `compute-and-ensure-slot-accessors` describes how slots are accessed

The standard method does a simple structure access

Object Oriented Languages

Metaobject Protocols

Structure: how are values to be stored in an object?

Most objects are implemented as structures with the object accessors implemented as simple structure references

The generic function `compute-and-ensure-slot-accessors` describes how slots are accessed

The standard method does a simple structure access

So if you want slots that live on disk rather than in memory, or count the number of times they are accessed, add a method here

Object Oriented Languages

Metaobject Protocols

Classes: The generic functions `allocate` and `initialize` describe how objects are created

Object Oriented Languages

Metaobject Protocols

Classes: The generic functions `allocate` and `initialize` describe how objects are created

So if you want to have an object that lives outside the normal class hierarchy, or has a different structure, add methods here

Object Oriented Languages

Metaobject Protocols

Classes: The generic functions `allocate` and `initialize` describe how objects are created

So if you want to have an object that lives outside the normal class hierarchy, or has a different structure, add methods here

And so on

Object Oriented Languages

Metaobject Protocols

There are standard, predefined, methods everywhere, of course, that do the “usual” things, i.e., they implement the behaviour you would expect from a normal OO language

Object Oriented Languages

Metaobject Protocols

There are standard, predefined, methods everywhere, of course, that do the “usual” things, i.e., they implement the behaviour you would expect from a normal OO language

For example, a method on `initialize` that fills in a newly allocated object

Object Oriented Languages

Metaobject Protocols

There are standard, predefined, methods everywhere, of course, that do the “usual” things, i.e., they implement the behaviour you would expect from a normal OO language

For example, a method on `initialize` that fills in a newly allocated object

But if you want to change how that happens, you can

Object Oriented Languages

Metaobject Protocols

Metaobject protocols are very powerful and so are easy to abuse

Object Oriented Languages

Metaobject Protocols

Metaobject protocols are very powerful and so are easy to abuse

And easy to misuse accidentally

Object Oriented Languages

Metaobject Protocols

Metaobject protocols are very powerful and so are easy to abuse

And easy to misuse accidentally

But they do allow you to do exactly what you want in an OO system

Object Oriented Languages

Metaobject Protocols

Metaobject protocols are very powerful and so are easy to abuse

And easy to misuse accidentally

But they do allow you to do exactly what you want in an OO system

You are not limited by the language, only your imagination

Object Oriented Languages

Metaobject Protocols

Exercise Investigate the `Metaobject` class in Java

Exercise Investigate the metaobject system in Python

Exercise Investigate Joose, the JavaScript metaobject system

Exercise Investigate Moose, the Perl metaobject system

Object Oriented Languages

End of OO

A note on efficiency of OO

Object Oriented Languages

End of OO

A note on efficiency of OO

You might think, with all this complexity, that OO languages must be very inefficient, and produce code that runs only very slowly

Object Oriented Languages

End of OO

A note on efficiency of OO

You might think, with all this complexity, that OO languages must be very inefficient, and produce code that runs only very slowly

The good thing is that with careful design of the language and good compilers an OO language need not be any less efficient than, say, a procedural language

Object Oriented Languages

End of OO

A note on efficiency of OO

You might think, with all this complexity, that OO languages must be very inefficient, and produce code that runs only very slowly

The good thing is that with careful design of the language and good compilers an OO language need not be any less efficient than, say, a procedural language

For example, Rust and C++ are pretty much as efficient as C

Object Oriented Languages

End of OO

This is because many of the complex OO mechanisms (method lookup, attribute lookup, etc.) have the potential of being statically done at compile time, so a method call can be as efficient as a function call

Object Oriented Languages

End of OO

This is because many of the complex OO mechanisms (method lookup, attribute lookup, etc.) have the potential of being statically done at compile time, so a method call can be as efficient as a function call

Perhaps the compiler is slower, but the generated code should not be

Object Oriented Languages

End of OO

This is because many of the complex OO mechanisms (method lookup, attribute lookup, etc.) have the potential of being statically done at compile time, so a method call can be as efficient as a function call

Perhaps the compiler is slower, but the generated code should not be

Dynamic languages can have a fair amount of runtime overhead, though

Object Oriented Languages

End of OO

This is because many of the complex OO mechanisms (method lookup, attribute lookup, etc.) have the potential of being statically done at compile time, so a method call can be as efficient as a function call

Perhaps the compiler is slower, but the generated code should not be

Dynamic languages can have a fair amount of runtime overhead, though

Of course, it's easy to design and implement an OO language poorly, just as any other feature, but there are few intrinsic reasons why OO should be slow

Object Oriented Languages

End of OO

Object Oriented languages and OO concepts are widely used

Object Oriented Languages

End of OO

Object Oriented languages and OO concepts are widely used

There are a large number of ways of doing OO

Object Oriented Languages

End of OO

Object Oriented languages and OO concepts are widely used

There are a large number of ways of doing OO

From no classes and no inheritance all the way to MOPs

Object Oriented Languages

End of OO

Object Oriented languages and OO concepts are widely used

There are a large number of ways of doing OO

From no classes and no inheritance all the way to MOPs

You need to able to make a choice!

Object Oriented Languages

End of OO

Object Oriented languages and OO concepts are widely used

There are a large number of ways of doing OO

From no classes and no inheritance all the way to MOPs

You need to able to make a choice!

Including choosing not to use OO

Object Oriented Languages

End of OO

C++ is history repeated as tragedy. Java is history repeated as farce
Scott McKay

Object-oriented programming is an exceptionally bad idea which could only have originated in California
Edsger Dijkstra

Object oriented programs are offered as alternatives to correct ones
Edsger Dijkstra

The End

There are many languages out there, both general purpose and specialist

The End

There are many languages out there, both general purpose and specialist

We have briefly covered many aspects of language design

The End

There are many languages out there, both general purpose and specialist

We have briefly covered many aspects of language design

And there are many more factors we could talk about in making choices of language

The End

Or language**s**

The End

Or language**s**

Often a single project can use several languages, each suited to its part of the project

The End

Or language**s**

Often a single project can use several languages, each suited to its part of the project

For example, using Python to manage a computation in C, while using Java to display the results graphically

The End

Or language**s**

Often a single project can use several languages, each suited to its part of the project

For example, using Python to manage a computation in C, while using Java to display the results graphically

And then the interoperability of languages becomes important

The End

Or language**s**

Often a single project can use several languages, each suited to its part of the project

For example, using Python to manage a computation in C, while using Java to display the results graphically

And then the interoperability of languages becomes important

For example, can you easily join together code written in Java and C?

The End

Or language**s**

Often a single project can use several languages, each suited to its part of the project

For example, using Python to manage a computation in C, while using Java to display the results graphically

And then the interoperability of languages becomes important

For example, can you easily join together code written in Java and C?

Or Java and Python?

The End

How does the managed (GC) nature of Java interact with the manual memory allocation in C? Or the GC in Java with the different GC in Python?

The End

How does the managed (GC) nature of Java interact with the manual memory allocation in C? Or the GC in Java with the different GC in Python?

Do types in the languages match? An integer in Python is represented in a different way to an integer in Java or C

The End

How does the managed (GC) nature of Java interact with the manual memory allocation in C? Or the GC in Java with the different GC in Python?

Do types in the languages match? An integer in Python is represented in a different way to an integer in Java or C

Sometimes integers and floating point numbers are represented in the same way, but what of Java's objects and C's structs?

The End

How does the managed (GC) nature of Java interact with the manual memory allocation in C? Or the GC in Java with the different GC in Python?

Do types in the languages match? An integer in Python is represented in a different way to an integer in Java or C

Sometimes integers and floating point numbers are represented in the same way, but what of Java's objects and C's structs?

Some languages specify a *foreign function interface* (FFI) that describes how to write code that interoperates with another language

The End

How does the managed (GC) nature of Java interact with the manual memory allocation in C? Or the GC in Java with the different GC in Python?

Do types in the languages match? An integer in Python is represented in a different way to an integer in Java or C

Sometimes integers and floating point numbers are represented in the same way, but what of Java's objects and C's structs?

Some languages specify a *foreign function interface* (FFI) that describes how to write code that interoperates with another language

Quite often a FFI to C, as C is very widely supported and used and is a "lowest common denominator" of languages

The End

Interopability is important in any moderately sized project and will affect which languages you may choose to use

The End

Interopability is important in any moderately sized project and will affect which languages you may choose to use

But it is important to remember the non-functional issues, too

The End

Interopability is important in any moderately sized project and will affect which languages you may choose to use

But it is important to remember the non-functional issues, too

What is the programming environment? E.g., quality compilers, IDEs, debuggers

The End

Interopability is important in any moderately sized project and will affect which languages you may choose to use

But it is important to remember the non-functional issues, too

What is the programming environment? E.g., quality compilers, IDEs, debuggers

What languages are you familiar with and happy programming in?

The End

A lot of languages support features that are designed to help the “novice programmer”

The End

A lot of languages support features that are designed to help the “novice programmer”

But programmers don’t stay novice, so what features are there to help the intermediate or advanced programmers?

The End

And there are the more niche considerations, e.g., energy usage can be very important in certain situations, for example mobile phones, or when you are thinking about green computing

The End

And there are the more niche considerations, e.g., energy usage can be very important in certain situations, for example mobile phones, or when you are thinking about green computing

How much energy does the final program take to run?

The End

And there are the more niche considerations, e.g., energy usage can be very important in certain situations, for example mobile phones, or when you are thinking about green computing

How much energy does the final program take to run?

If your program is going to run very many times, or you need to worry about battery life, then this measure is important

The End

And there are the more niche considerations, e.g., energy usage can be very important in certain situations, for example mobile phones, or when you are thinking about green computing

How much energy does the final program take to run?

If your program is going to run very many times, or you need to worry about battery life, then this measure is important

We touched on this briefly when talking about bytecoding with JIT and AOT and other execution approaches

The End

And there are the more niche considerations, e.g., energy usage can be very important in certain situations, for example mobile phones, or when you are thinking about green computing

How much energy does the final program take to run?

If your program is going to run very many times, or you need to worry about battery life, then this measure is important

We touched on this briefly when talking about bytecoding with JIT and AOT and other execution approaches

Experiments show that C programs generally take the least energy to run

The End

The Top 10 performers:

C	1.00
Rust	1.03
C++	1.34
Ada	1.70
Java	1.98
Pascal	2.14
Chapel	2.18
Lisp	2.27
Ocaml	2.40
Fortran	2.52

Relative energy uses of programs written in various languages

The End

The Top 10 performers:

C	1.00
Rust	1.03
C++	1.34
Ada	1.70
Java	1.98
Pascal	2.14
Chapel	2.18
Lisp	2.27
Ocaml	2.40
Fortran	2.52

Relative energy uses of programs written in various languages

Python is number 26 at 75.88

The End

Exercise Read “Energy Efficiency across Programming Languages: How Do Energy, Time, and Memory Relate?” by Pereira et. al., SLE 2017: Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, October 2017, pp. 256-267, `https://doi.org/10.1145/3136014.3136031`

The End

Do you need support for modern character sets (different languages, emoji)?

The End

Do you need support for modern character sets (different languages, emoji)?

Support for Unicode is patchy in some languages, good in others

The End

Do you need support for modern character sets (different languages, emoji)?

Support for Unicode is patchy in some languages, good in others

And some languages (and operating) systems use different encodings for the characters

The End

Do you need support for modern character sets (different languages, emoji)?

Support for Unicode is patchy in some languages, good in others

And some languages (and operating) systems use different encodings for the characters

For example, The Web mostly uses UTF-8 encodings, whereas SMS text messages use UTF-16 and Java, JavaScript and Windows are stuck using UTF-16 internally

The End

Do you need support for modern character sets (different languages, emoji)?

Support for Unicode is patchy in some languages, good in others

And some languages (and operating) systems use different encodings for the characters

For example, The Web mostly uses UTF-8 encodings, whereas SMS text messages use UTF-16 and Java, JavaScript and Windows are stuck using UTF-16 internally

Rust, Go and Python use UTF-8

The End

Exercise What does C do?

Exercise Find out what your favourite languages do

The End

Next, what is the support for writing documentation?

The End

Next, what is the support for writing documentation?

Tools like Sphinx for Python, Doxygen for C++, Godoc for Go, Rustdoc for Rust and so on

The End

Next, what is the support for writing documentation?

Tools like Sphinx for Python, Doxygen for C++, Godoc for Go, Rustdoc for Rust and so on

Exercise Read about “Literate programming” by Knuth

The End

Next, what is the support for writing documentation?

Tools like Sphinx for Python, Doxygen for C++, Godoc for Go, Rustdoc for Rust and so on

Exercise Read about “Literate programming” by Knuth

Hint Whenever you fix a bug in your code, put a comment on it!

The End

Exercise Which in the more helpful comment here?

```
// set c to 0
c = 0;

// initialize the object count
c = 0;
```

The End

Exercise Which in the more helpful comment here?

```
// set c to 0
c = 0;
```

```
// initialize the object count
c = 0;
```

And which of the above two kinds of comment appears most in student code?

The End

And working in teams

The End

And working in teams

When joining a team, what languages are they already using?

The End

And working in teams

When joining a team, what languages are they already using?

What code management tools are they using?

The End

And working in teams

When joining a team, what languages are they already using?

What code management tools are they using?

What does your boss tell you to do?

The End

And so on

The End

And so on

Picking the right language(s) can be complicated: but it will repay you well to think carefully about this

The End

Remember:

- Not all OO languages have classes
- Not all OO languages have polymorphism
- Not all class-based languages have inheritance
- OO languages are not necessarily slow or fast
- OO languages are not necessarily easy or hard to use
- Procedural languages are not necessarily slow or fast
- Procedural languages are not necessarily easy or hard to use
- Functional languages are not necessarily slow or fast
- Functional languages are not necessarily easy or hard to use

The End

- Not all bytecode compilers are fast
- Not all bytecode compilers are slow
- Not all native compilers are fast
- Not all native compilers are slow
- Not all VMs are large
- Not all VMs are compact

The End

Exercise Find examples of languages that confirm each of these statements (e.g., original JavaScript did not have classes)

The End

Exercise Consider:

- procedural, OO, etc.: flow of execution is determined by the code
- data driven: flow is determined by the data
- declarative: flow is determined by the system
- event: flow is determined by the events

The End

If anyone tells you that one language is better than another, you will know you are a better programmer than they are

The End

If anyone tells you that one language is better than another, you will know you are a better programmer than they are

It's not a case that one language is better than another, more that one is *better for the problem in hand*

The End

If anyone tells you that one language is better than another, you will know you are a better programmer than they are

It's not a case that one language is better than another, more that one is *better for the problem in hand*

Be aware and learn the *concepts*, they are transferable between many languages

The End

If anyone tells you that one language is better than another, you will know you are a better programmer than they are

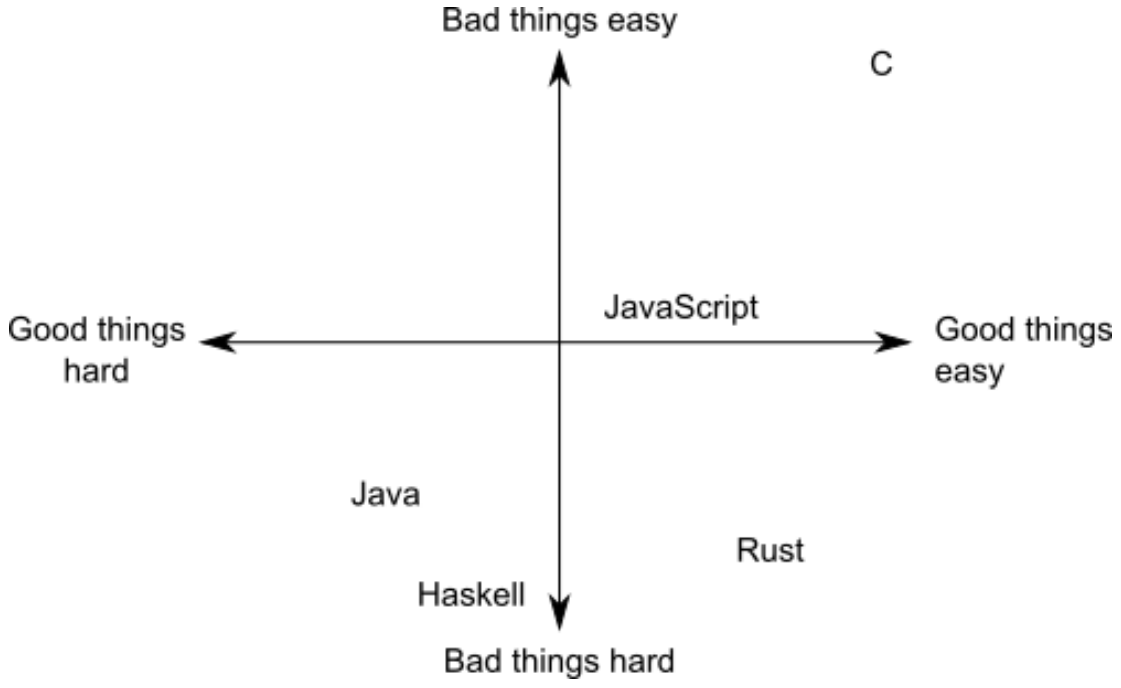
It's not a case that one language is better than another, more that one is *better for the problem in hand*

Be aware and learn the *concepts*, they are transferable between many languages

Pick the right tool for the job

The End

Exercise Add more languages to this chart:



Programming good and bad, hard or easy

The End

Exercise For as many languages as you can, classify them as

general purpose; specific purpose; procedural; logic; declarative; imperative; functional; macro; scripting; event driven; simulation; dataflow; markup; parallel; GC; manual memory management; other memory management; typed; untyped; strongly typed; weakly typed; statically typed; dynamically typed; manifest typed; implicit typed; polymorphic typed; has overloading; has higher kinded or rank types; has dependent types; constant variables; call by value; call by reference; call by name; call by need; interpreted; compiled; byte compiled; managed code or data; unmanaged code or data; expression based; statement based; error handing support; object oriented; class centred OO; object centred OO; single inheritance; multiple inheritance; traits/mixins/interfaces; object receiver; generic functions; single dispatch; multiple dispatch; prototyping; delegation; has metaobjects; has a good development environment; has a good debugger; is easy to learn

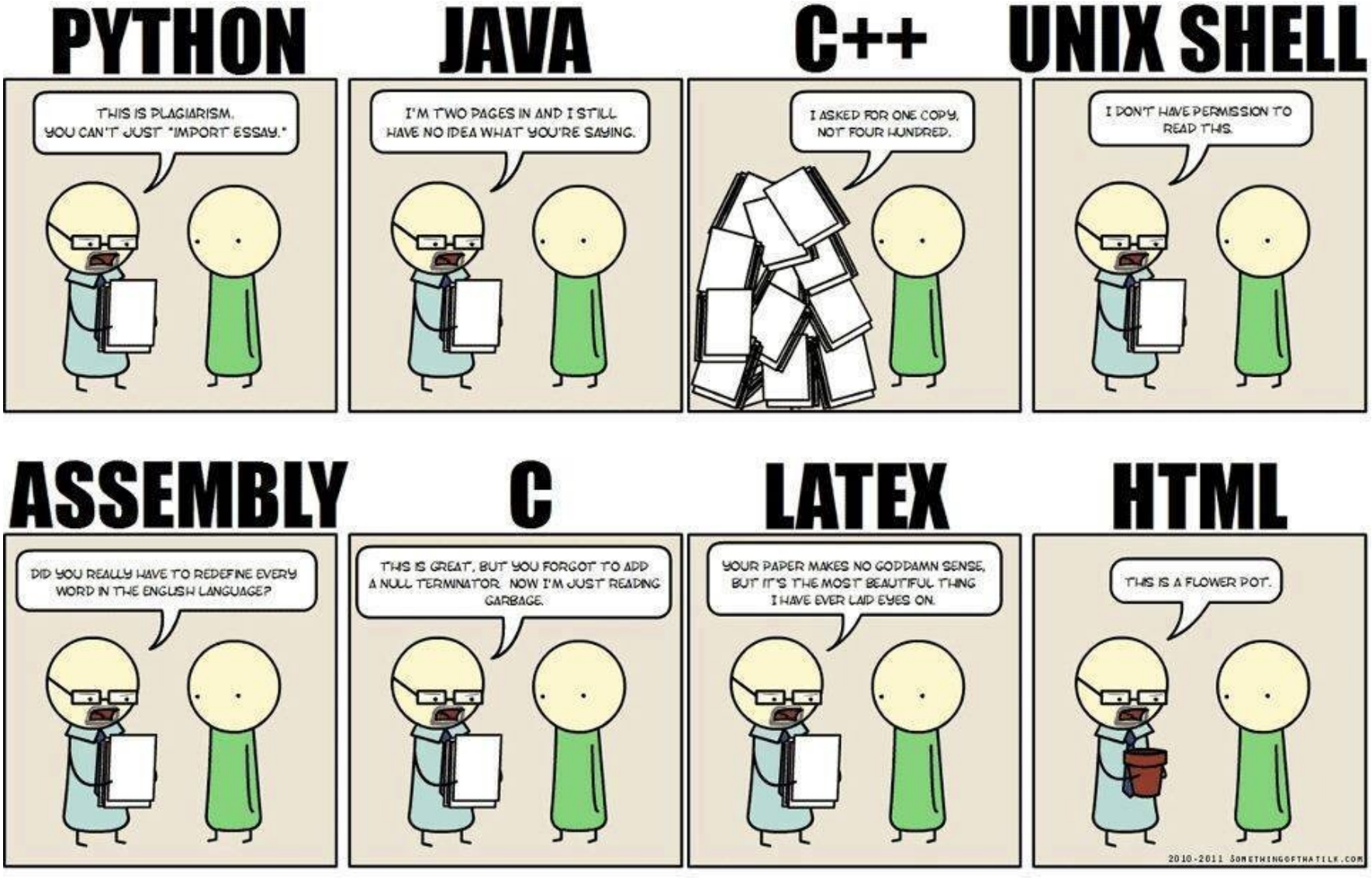
The End

> *There are only two kinds of languages: the ones people complain about and the ones nobody uses*

Bjarne Stroustrup, creator of C++

> *There are only two kinds of C++: the language subsets that Stroustrup insists that people should use, and the subsets that people actually use*

Anon



(Source unknown)

The End

How to shoot yourself in the foot while programming:

The End

How to shoot yourself in the foot while programming:

Pick the wrong tools

A# .NET A# (Axiom) A-0 System A+ A++ ABAP ABC ABC ALGOL ABSET ABSYS ACC Accent Ace DASL ACL2 Avicsoft ACT-III Action! ActionScript Ada Adenine Agda Agilent VEE Agora AIMMS Alef ALF ALGOL 58 ALGOL 60 ALGOL 68 ALGOL W Alice Alma-0 AmbientTalk Amiga E AMOS AMPL AngularJS Apex (Salesforce.com) APL App Inventor for Android's visual block language AppleScript Arc ARexx Argus AspectJ Assembly language ATS Ateji PX AutoHotkey Autocoder AutoIt AutoLISP / Visual LISP Averest AWK Axum Active Server Pages ASP.NET B Babbage Bash BASIC bc BCPL BeanShell Batch (Windows/Dos) Bertrand BETA Bigwig Bistro BitC BLISS Blockly BlooP Blue Boo Boomerang Bourne shell (including bash and ksh) BREW BPEL Business Basic C C– C++ C# C/AL Caché ObjectScript C Shell Caml Cayenne CDuce Cecil Cesil Céu Ceylon CFEngine CFML Cg Ch Chapel Charity Charm Chef CHILL CHIP-8 chomski ChucK CICS Cilk Citrine CL (IBM) Claire Clarion Clean Clipper CLIPS CLIST Clojure CLU CMS-2 COBOL CobolScript Cobra CODE CoffeeScript ColdFusion COMAL Combined Programming Language (CPL) COMIT Common Intermediate Language (CIL) Common Lisp (CL) COMPASS Component Pascal Constraint Handling Rules (CHR) COMTRAN Converge Cool Coq Coral 66 Corn CorVision COWSEL CPL CPL Cryptol csh Csound CSP CUDA Curl Curry Cybil Cyclone Cython D DASL Dart DataFlex Datalog DATATRIEVE dBase dc DCL Deesel (formerly G) Delphi DinkC DIBOL Dog Draco DRAKON Dylan DYNAMO E E# EarSketch Ease Easy PL/I Easy Programming Language EASYTRIEVE PLUS ECMAScript Edinburgh IMP EGL Eiffel ELAN Elixir Elm Emacs Lisp Emerald Epigram EPL Erlang es Escher ESPOL Esterel Etoys Euclid Euler Euphoria EusLisp Robot Programming Language CMS EXEC EXEC 2 Executable UML F F# Factor Falcon Fantom FAUST FFP Fjölnir FL Flavors Flex FlooP FLOW-MATIC FOCAL FOCUS FOIL FORMAC @Formula Forth Fortran Fortress FoxBase FoxPro FP FPr Franz Lisp Frege F-Script G Game Maker Language GameMonkey Script GAMS GAP G-code Genie GDL GJ GEORGE GLSL GNU E GM Go Go! GOAL Gödel Godiva Golo GOM (Good Old Mad) Google Apps Script Gosu GOTRAN GPSS GraphTalk GRASS Groovy Hack Hadoop HAGGIS HAL/S Hamilton C shell Harbour Hartmann pipelines Haskell Haxe Hermes High Level Assembly HLSL Hop Hopscotch Hope Hugo Hume HyperTalk IBM Basic assembly language IBM HAScript IBM Informix-4GL IBM RPG ICI Icon Id IDL Idris IMP Inform INTERLISP Io Ioke IPL IPTSCRAE ISLISP ISPF ISWIM J J# J++ JADE Jako JAL Janus JASS Java JavaScript JCL JEAN Join Java JOSS Joule JOVIAL Joy JScript JScript .NET JavaFX Script Julia Jython K Kaleidoscope Karel Karel++ KEE Kixtart Klerer-May System KIF Kojo Kotlin KRC KRL KRL KRYPTON ksh L L# .NET LabVIEW Ladder Lagoona LANSA Lasso Lava LC-3 Leda Legoscript LIL LilyPond Limbo Limnor LINC Lingo LIS LISA Lisaac Lisp Lite-C Lithe Little b Logo Logtalk LotusScript LPC LSE LSL LiveScript Lua Lucid Lustre LYaPAS Lynx M2001 M4 M# Machine code MAD (Michigan Algorithm Decoder) MAD/I Magik Magma make Maple MAPPER now part of BIS MARK-IV now VISION:BUILDER Mary MASM Microsoft Assembly x86 MATH-MATIC Mathematica MATLAB Maxima (see also Macsyma) Max (Max Msp) MaxScript internal language 3D Studio Max Maya (MEL) MDL Mercury Mesa Metafont Microcode MicroScript MIIS Milk MIMIC Mirah Miranda MIVA Script ML Model 204 Modelica Modula Modula-2 Modula-3 Mohol MOO Mortran Mouse MPD Mathcad MSIL MSL MUMPS Mystic Programming Language (MPL) NASM Napier88 Neko Nemerle nesC NESL Net.Data NetLogo NetRexx NewLISP NEWP Newspeak NewtonScript NGL Nial Nice Nickle Nim NO NPL Not eXactly C (NXC) Not Quite C (NQC) NSIS Nu Numpy NWScript NXT-G o:XML Oak Oberon OBJ2 Object Lisp ObjectLOGO Object REXX Object Pascal Objective-C Objective-J Obliq OCaml occam occam-π Octave OmniMark Onyx Opa Opal OpenCL OpenEdge ABL OPL OpenVera OPS5 OptimJ Orc ORCA/Modula-2 Oriel Orwell Oxygene Oz P" P# ParaSail (programming language) PARI/GP Pascal PCASTL PCF PEARL PeopleCode Perl PDL Perl 6 Pharo PHP Phrogram Pico Picolisp Pict Pike PIKT PILOT Pipelines Pizza PL-11 PL/0 PL/B PL/C PL/I PL/M PL/P PL/SQL PL360 PLANC Plankalkül Planner PLEX PLEXIL Plus POP-11 POP-2 PostScript PortablE Powerhouse PowerBuilder PowerShell PPL Processing Processing.js Prograph PROIV Prolog PROMAL Promela PROSE PROTEL ProvideX Pro*C Pure Pure Data Python Q Qalb QtScript QuakeC QPL R R++ Racket RAPID Rapira Ratfiv Ratfor rc REBOL Red Redcode REFAL Reia REXX Rlab ROOP RPG RPL RSL RTL/2 Ruby RuneScript Rust S S2 S3 S-Lang S-PLUS SA-C SabreTalk SAIL SALSA SAM76 SAS SASL Sather Sawzall SBL Scala Scheme Scilab Scratch Script.NET Sed Seed7 Self SenseTalk SequenceL SETL SIMPOL SIGNAL SiMPLE SIMSCRIPT Simula Simulink Singularity SISAL SLIP SMALL Smalltalk Small Basic SML Strongtalk Snap! SNOBOL(SPITBOL) Snowball SOL Span SPARK Speedcode SPIN SP/k SPS SQL SQR Squeak Squirrel SR S/SL Stackless Python Starlogo Strand Stata Stateflow Subtext SuperCollider SuperTalk Swift SYMPL SyncCharts SystemVerilog T TACL TACPOL TADS TAL Tcl Tea TECO TELCOMP TeX TEX TIE Timber TMG Tom TOM TouchDevelop Toi Topspeed TPU Trac TTM T-SQL Transcript TTCN Turing TUTOR TXL TypeScript Turbo C++ Ubercode UCSD Pascal Umple Unicon Uniface UNITY Unix shell UnrealScript Vala Verilog VHDL Visual Basic Visual Basic .NET Visual DataFlex Visual DialogScript Visual Fortran Visual FoxPro Visual J++ Visual J# Visual Objects Visual Prolog VSXu vvvv WATFIV WATFOR WebDNA WebQL Whiley Windows PowerShell Winbatch Wolfram Language Wyvern X++ X# X10 XBL XC xHarbour XL Xojo XOTcl XPL XPL0 XQuery XSB XSharp XSLT XPath Xtend Yorick YQL Yoix Z notation Zeno ZOPL Zsh