What is the difference between statically typed and dynamically typed languages?

Asked 10 years, 10 months ago Active 7 months ago Viewed 430k times

965

393

I hear a lot that new programming languages are dynamically typed but what does it actually mean when we say a language is dynamically typed vs. statically typed?

programming-languages computer-science static-typing dynamic-typing

edited Jul 15 '19 at 9:20

SirSaleh

1,051 ● 2 ● 18 ● 33

asked Oct 4 '09 at 22:36

Rachel

86.4k ● 110 ● 249 ● 358

25 Good question but your accepted answer is not the correct answer. – J D Oct 26 '13 at 8:26

43 can you pls refer something which is correct then ? – Sagiruddin Mondal Jun 10 '15 at 8:03

@EricLeschinski I think unit tests help now with that problem and dynamically typed languages such as **JavaScript** can be coded with reassurance that it will hold up, thus making it eligible for enterprise software development, wouldn't you think? – pixel 67 Jan 11 '16 at 2:58

8 At best, those unit tests deteriorate with time and get turned off by co-workers trying to increase job security, at worst, they never get written in the first place. It's like suggesting to a pro mechanic to use duct tape on his customer's cars. Yes junior, using duct tape on this transmission job is a good idea... for you. – Eric Leschinski Jan 11 '16 at 3:23

3 I know this question is 10 years old, but from what I read in the comments, you should probably switch to accept the answer from "Christopher Tokar". – Rev1.0 Feb 7 '19 at 9:35

16 Answers

Active Oldest Votes

871

Statically typed languages

A language is statically typed if the type of a variable is known at compile time. For some languages this means that you as the programmer must specify what type each variable is (e.g.: Java, C, C++); other languages offer some form of *type inference*, the

capability of the type system to deduce the type of a variable (e.g.: OCaml, Haskell, Scala, Kotlin)

The main advantage here is that all kinds of checking can be done by the compiler, and therefore a lot of trivial bugs are caught at a very early stage.

Examples: C, C++, Java, Rust, Go, Scala

Dynamically typed languages

A language is dynamically typed if the type is associated with run-time values, and not named variables/fields/etc. This means that you as a programmer can write a little quicker because you do not have to specify types every time (unless using a statically-typed language with *type inference*).

Examples: Perl, Ruby, Python, PHP, JavaScript

Most scripting languages have this feature as there is no compiler to do static type-checking anyway, but you may find yourself searching for a bug that is due to the interpreter misinterpreting the type of a variable. Luckily, scripts tend to be small so bugs have not so many places to hide.

Most dynamically typed languages do allow you to provide type information, but do not require it. One language that is currently being developed, Rascal, takes a hybrid approach allowing dynamic typing within functions but enforcing static typing for the function signature.

edited Jan 7 '19 at 23:04

GG.

16.2k ● 9 ● 65 ● 111

answered Oct 4 '09 at 23:11

NomeN

14.5k ● 6 ● 29 ● 33

6 @NomeN Can you name any dynamically typed language which implements HM type inference? – Pete Kirkham Oct 5 '09 at 10:05

88 "A language is dynamically typed if the type of a variable is interpreted at runtime": No. A language is dynamically typed if the type is associated with run-time values, and not named variables/fields/etc. – Paul Biggar Oct 5 '09 at 10:48

15 Incorrect, static typing means "that a reference value is manifestly (which is not the same as at compile time) constrained with respect to the type of the value it can denote, and that the language implementation, whether it is a compiler or an interpreter, both enforces and uses these constraints as much as possible." cited from : c2.com/cgi/wiki?StaticTyping which, how I understand it, is correct. – Matthias Wolf Jan 24 '13 at 1:43

6 The thing most obvious about the type systems of Java, C, C++, Pascal, and many other widely-used “industry” languages is not that they are statically typed, but that they are explicitly typed.In other words, they require lots of type declarations. (In the world of less explicitly typed languages, where these declarations are optional, they are often called “type annotations”.) This has nothing to do with static types. continued.. – Vipresh Aug 16 '14 at 10:27

7 The first statically typed languages were explicitly typed by necessity. However, type inference algorithms – techniques for looking at source code

with no type declarations at all, and deciding what the types of its variables are have existed for many years now. The ML language, which uses it. Haskell, which improves on it, is now about 15 years old. Even C# is now adopting the idea, which will raise a lot of eyebrows and undoubtedly give rise to claims of its being “weakly typed”. continued... – Vipresh Aug 16 '14 at 10:29

406

Statically typed programming languages do type checking (i.e. the process of verifying and enforcing the constraints of types) at *compile-time* as opposed to *run-time*.

Dynamically typed programming languages do type checking at *run-time* as opposed to *compile-time*.

Examples of statically typed languages are :- Java, C, C++

Examples of dynamically typed languages are :- Perl, Ruby, Python, PHP, JavaScript

edited Dec 23 '19 at 11:53

Sreekanth Karumanaghat
3,089 ● 5 ● 33 ● 64

answered Oct 5 '09 at 14:12

Christopher Tokar
10k ● 8 ● 34 ● 56

19 I think this is the best answer. In particular, the accepted answer is largely factually incorrect. – J D Oct 26 '13 at 8:22

1 @JonHarrop In what ways specifically? – 1252748 Dec 8 '14 at 0:20

21 @thomas: "This means that you as a programmer can write a little quicker because you do not have to specify type everytime". You do not have to specify the type every time with static typing if you have type inference. See SML, OCaml, F#, Haskell... – J D Dec 8 '14 at 22:06

1 In statically typed prog languages the type checking is done before runtime, but not exactly at compile time. – Luiz Felipe Nov 24 '17 at 14:21

312

Here is an example contrasting how Python (dynamically typed) and Go (statically typed) handle a type error:

```
def silly(a):
    if a > 0:
        print 'Hi'
    else:
        print 5 + '3'
```

Python does type checking at run time, and therefore:

```
silly(2)
```

Runs perfectly fine, and produces the expected output `Hi`. Error is only raised if the problematic line is hit:

```
silly(-1)
```

Produces

> ```
> TypeError: unsupported operand type(s) for +: 'int' and 'str'
> ```

because the relevant line was actually executed.

Go on the other hand does type-checking at compile time:

```
package main

import ("fmt"
)

func silly(a int) {
    if (a > 0) {
        fmt.Println("Hi")
    } else {
        fmt.Println("3" + 5)
    }
}

func main() {
    silly(2)
}
```

The above will not compile, with the following error:

> ```
> invalid operation: "3" + 5 (mismatched types string and int)
> ```

edited Dec 5 '17 at 0:58

John Kugelman

289k ● 61 ● 451 ● 501

answered Jan 6 '15 at 2:49

Akavall

60.7k ● 37 ● 170 ● 210

5 Thanks for the tidy example. So I infer that all of the scripting languages are Dynamically typed, as they are not compiled? – CʰAZ Sep 13 '15 at 12:50

8 yes. all scripting languages are dynamically typed, because their is no compiler to do static type checking anyway. This point has been illustrated in this article sitepoint.com/typing-versus-dynamic-typing . – Shashi Nov 19 '15 at 8:20

9 Scala can be used as scripting language and it is statically typed! #discussion @Shashi – Sagiruddin Mondal Jan 2 '16 at 9:53

3 @Shashi Compilation doesn't mean statically-typed. Haskell can be interpreted with `runhaskell`, for example. – BalinKingOfMoria Reinstate CMs Feb 13 '16 at 15:07

2 Also Scripting language does NOT mean interpreted language. TypeScript is statically typed, compiled/transpiled, but scripting language. – metalim Aug 26 '16 at 10:13

162

Simply put it this way: in a **statically typed language** variables' types are *static*, meaning once you set a variable to a type, you cannot change it. That is because typing is associated with the variable rather than the value it refers to.

For example in Java:

```
String str = "Hello";  //variable str statically typed as string
str = 5;               //would throw an error since str is supposed to be a string only
```

Where on the other hand: in a **dynamically typed language** variables' types are *dynamic*, meaning after you set a variable to a type, you CAN change it. That is because typing is associated with the value it assumes rather than the variable itself.

For example in Python:

```
str = "Hello" # variable str is linked to a string value
str = 5       # now it is linked to an integer value; perfectly OK
```

So, it is best to think of variables in dynamically typed languages as *just generic pointers* to typed values.

To sum up, **type** describes (or should have described) the variables in the language rather than the language itself. It could have been better used as **a language with statically typed variables** versus **a language with dynamically typed variables** IMHO.

Statically typed languages are generally compiled languages, thus, the compilers check the types (make perfect sense right? as types are not allowed to be changed later on at run time).

Dynamically typed languages are generally interpreted, thus type checking (if any) happens at run time when they are used. This of course brings some performance cost, and is one of the reasons dynamic languages (e.g., python, ruby, php) do not scale as good as the typed ones (java, c#, etc.). From another perspective, statically typed languages have more of a start-up cost: makes you usually write more code, harder code. But that pays later off.

The good thing is both sides are borrowing features from the other side. Typed languages are incorporating more dynamic features, e.g., generics and dynamic libraries in c#, and dynamic languages are including more type checking, e.g., type annotations in python, or HACK variant of PHP, which are usually not core to the language and usable on demand.

When it comes to technology selection, neither side has an intrinsic superiority over the other. It is just a matter of preference whether you want more control to begin with or flexibility. just pick the right tool for the job, and make sure to check what is available in terms of the opposite before considering a switch.

edited May 5 '17 at 14:10

answered Nov 30 '15 at 17:28

mehmet

5,185 ● 3 ● 26 ● 39

9 This makes a lot of sense. I think it explains at least the reasoning behind the names much better than other answers here. – JamEngulfer Feb 24 '16 at 11:37

According to this reference Python is both a statically typed and dynamically typed language: wiki.python.org/moin/… Does anyone know why? – modulitos Apr 25 '16 at 5:12

1 Lucas, To the contrary the document keeps iterating that Python is both strongly and dynamically typed. Where did you see that? Can you quote? – mehmet Apr 26 '16 at 20:00

2 I think this answer best communicates the the concept in the simplest fashion. A lot of other answers try to abstractly describe the concept, but fail at some detail. I would rather see this answer at the top of the list. – Hawkeye Feb 28 '17 at 23:39

5 Most other answers created more questions in my mind. This one cleared all of them. This answer really should be at the top IMHO – Hami Torun Apr 29 '17 at 8:05

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

39

Static typing

A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time. In static typing, types are associated with variables not values. Statically typed languages include Ada, C, C++, C#, JADE, Java, Fortran, Haskell, ML, Pascal, Perl (with respect to distinguishing scalars, arrays, hashes and subroutines) and Scala. Static

typing is a limited form of program verification (see type safety): accordingly, it allows many type errors to be caught early in the development cycle. Static type checkers evaluate only the type information that can be determined at compile time, but are able to verify that the checked conditions hold for all possible executions of the program, which eliminates the need to repeat type checks every time the program is executed. Program execution may also be made more efficient (i.e. faster or taking reduced memory) by omitting runtime type checks and enabling other optimizations.

Because they evaluate type information during compilation, and therefore lack type information that is only available at run-time, static type checkers are conservative. They will reject some programs that may be well-behaved at run-time, but that cannot be statically determined to be well-typed. For example, even if an expression always evaluates to true at run-time, a program containing the code

```
if <complex test> then 42 else <type error>
```

will be rejected as ill-typed, because a static analysis cannot determine that the else branch won't be taken.[1] The conservative behaviour of static type checkers is advantageous when evaluates to false infrequently: A static type checker can detect type errors in rarely used code paths. Without static type checking, even code coverage tests with 100% code coverage may be unable to find such type errors. Code coverage tests may fail to detect such type errors because the combination of all places where values are created and all places where a certain value is used must be taken into account.

The most widely used statically typed languages are not formally type safe. They have "loopholes" in the programming language specification enabling programmers to write code that circumvents the verification performed by a static type checker and so address a wider range of problems. For example, Java and most C-style languages have type punning, and Haskell has such features as unsafePerformIO: such operations may be unsafe at runtime, in that they can cause unwanted behaviour due to incorrect typing of values when the program runs.

Dynamic typing

A programming language is said to be dynamically typed, or just 'dynamic', when the majority of its type checking is performed at run-time as opposed to at compile-time. In dynamic typing, types are associated with values not variables. Dynamically typed languages include Groovy, JavaScript, Lisp, Lua, Objective-C, Perl (with respect to user-defined types but not built-in types), PHP, Prolog, Python, Ruby, Smalltalk and Tcl. Compared to static typing, dynamic typing can be more flexible (e.g. by allowing programs to generate types and functionality based on run-time data), though at the expense of fewer a priori guarantees. This is because a dynamically typed language accepts and attempts to execute some programs which may be ruled as invalid by a static type checker.

Dynamic typing may result in runtime type errors—that is, at runtime, a value may have an unexpected type, and an operation nonsensical for that type is applied. This operation may occur long after the place where the programming mistake was made—that is, the place where the wrong type of data passed into a place it should not have. This makes the bug difficult to locate.

> Dynamically typed language systems, compared to their statically typed cousins, make fewer "compile-time" checks on the source code (but will check, for example, that the program is syntactically correct). Run-time checks can potentially be more sophisticated, since they can use dynamic information as well as any information that was present during compilation. On the other hand, runtime checks only assert that conditions hold in a particular execution of the program, and these checks are repeated for every execution of the program.
>
> Development in dynamically typed languages is often supported by programming practices such as unit testing. Testing is a key practice in professional software development, and is particularly important in dynamically typed languages. In practice, the testing done to ensure correct program operation can detect a much wider range of errors than static type-checking, but conversely cannot search as comprehensively for the errors that both testing and static type checking are able to detect. Testing can be incorporated into the software build cycle, in which case it can be thought of as a "compile-time" check, in that the program user will not have to manually run such tests.
>
> **References**
>
> 1. Pierce, Benjamin (2002). Types and Programming Languages. MIT Press. ISBN 0-262-16209-1.

edited Jun 20 at 9:12
Community ♦
1 ● 1

answered Oct 4 '09 at 22:37

Jacob
32.5k ● 12 ● 101 ● 157

75 The main idea of SO is to build a body of knowledge, not to provide links to other places. You should try to at least make an excerpt of the wiki that answers the question. – NomeN Oct 4 '09 at 23:13

5 It just seemed redundant since it's a link to wikipedia and not some transient website, but I'll remember that next time. – Jacob Oct 5 '09 at 13:24

2 somehow I still cannot think of an example in a dynamically typed language where a type is not clear at compile time but must be figured out at runtime. Could you provide me with some? – Novellizator Apr 18 '14 at 23:54

3 @Novellizator Old comment but imagine a scenario where some data is picked up from a remote server then that data is used to pick a property off of an object. Ex: `myObject[remoteDataName]`. Then there's no way of knowing which property it will pick or even if it's a valid property at all. – Mike Cluck Aug 12 '14 at 14:36

14

The terminology "dynamically typed" is unfortunately misleading. All languages are statically typed, and types are properties of expressions (not of values as some think). However, some languages have only one type. These are called uni-typed languages. One example of such a language is the untyped lambda calculus.

In the untyped lambda calculus, all terms are lambda terms, and the only operation that can be performed on a term is applying it to another term. Hence all operations always result in either infinite recursion or a lambda term, but never signal an error.

However, were we to augment the untyped lambda calculus with primitive numbers and arithmetic operations, then we could perform nonsensical operations, such adding two lambda terms together: `(λx.x) + (λy.y)`. One could argue that the only sane thing to do is to signal an error when this happens, but to be able to do this, each value has to be tagged with an indicator that indicates whether the term is a lambda term or a number. The addition operator will then check that indeed both arguments are tagged as numbers, and if they aren't, signal an error. Note that these tags are *not* types, because types are properties of programs, not of values produced by those programs.

A uni-typed language that does this is called dynamically typed.

Languages such as JavaScript, Python, and Ruby are all uni-typed. Again, the `typeof` operator in JavaScript and the `type` function in Python have misleading names; they return the tags associated with the operands, not their types. Similarly, `dynamic_cast` in C++ and `instanceof` in Java do *not* do type checks.

edited Oct 12 '15 at 17:02

answered Oct 12 '15 at 16:57

rightfold

28.7k ● 8 ● 79 ● 103

9

Compiled vs. Interpreted

"When source code is translated"

- **Source Code**: Original code (usually typed by a human into a computer)
- **Translation**: Converting source code into something a computer can read (i.e. machine code)
- **Run-Time**: Period when program is executing commands (after compilation, if compiled)
- **Compiled Language**: Code translated before run-time
- **Interpreted Language**: Code translated on the fly, during execution

---

Typing

"When types are checked"

`5 + '3'` is an example of a type error in *strongly typed* languages such as Go and Python, because they don't allow for "type coercion" -> the ability for a value to change type in certain contexts such as merging two types. *Weakly typed* languages, such as JavaScript, won't throw a type error (results in `'53'`).

- **Static**: Types checked before run-time

- **Dynamic**: Types checked on the fly, during execution

The definitions of "Static & Compiled" and "Dynamic & Interpreted" are quite similar...but remember it's "when types are checked" vs. "when source code is translated".

You'll get the same type errors irrespective of whether the language is compiled or interpreted! You need to separate these terms conceptually.

Python Example

Dynamic, Interpreted

```
def silly(a):
    if a > 0:
        print 'Hi'
    else:
        print 5 + '3'

silly(2)
```

Because Python is both interpreted and dynamically typed, it only translates and type-checks code it's executing on. The `else` block never executes, so `5 + '3'` is never even looked at!

What if it was statically typed?

A type error would be thrown before the code is even run. It still performs type-checking before run-time even though it is interpreted.

What if it was compiled?

The `else` block would be translated/looked at before run-time, but because it's dynamically typed it wouldn't throw an error! Dynamically typed languages don't check types until execution, and that line never executes.

Go Example

Static, Compiled

```
package main
```

```
import ("fmt"
)

func silly(a int) {
  if (a > 0) {
      fmt.Println("Hi")
  } else {
      fmt.Println("3" + 5)
  }
}

func main() {
  silly(2)
}
```

The types are checked before running (static) and the type error is immediately caught! The types would still be checked before run-time if it was interpreted, having the same result. If it was dynamic, it wouldn't throw any errors even though the code would be looked at during compilation.

Performance

A compiled language will have better performance at run-time if it's statically typed (vs. dynamically); knowledge of types allows for machine code optimization.

Statically typed languages have better performance at run-time intrinsically due to not needing to check types dynamically while executing (it checks before running).

Similarly, compiled languages are faster at run time as the code has already been translated instead of needing to "interpret"/translate it on the fly.

Note that both compiled and statically typed languages will have a delay before running for translation and type-checking, respectively.

More Differences

Static typing catches errors early, instead of finding them during execution (especially useful for long programs). It's more "strict" in that it won't allow for type errors anywhere in your program and often prevents variables from changing types, which further defends against unintended errors.

```
num = 2
num = '3' // ERROR
```

Dynamic typing is more flexible, which some appreciate. It typically allows for variables to change types, which can result in unexpected errors.

edited Dec 8 '17 at 21:48

answered Dec 8 '17 at 21:25

JBallin

3,130 ● 20 ● 32

"*Because Python is both interpreted and dynamically typed, it only translates and type-checks code it's executing on*" – that's not really the case. Python (at least the reference implementation) **compiles** all your code at import time (you can also compile modules before/without importing them). The compiler introduces various optimisations (at least as far as Python's dynamic nature allows). – Eli Korvigo Feb 5 '18 at 21:11

6

Statically typed languages: each variable and expression is already known at compile time.

(`int a;` a can take only integer type values at runtime)

Examples: C, C++, Java

Dynamically typed languages: variables can receive different values at runtime and their type is defined at run time.

(`var a;` a can take any kind of values at runtime)

Examples: Ruby, Python.

edited Nov 19 '18 at 8:29

Pang

8,127 ● 16 ● 73 ● 111

answered Oct 1 '17 at 14:35

Raman Gupta

934 ● 9 ● 7

5

Statically typed languages type-check at compile time and the type can NOT change. (Don't get cute with type-casting comments, a new variable/reference is created).

Dynamically typed languages type-check at run-time and the type of an variable CAN be changed at run-time.

edited Oct 18 '16 at 8:09

answered Jul 27 '16 at 6:56


SuB
1,440 ● 2 ● 16 ● 32


RESTfulGeoffrey
368 ● 3 ● 8

4

Sweet and simple definitions, but fitting the need: Statically typed languages binds the type to a variable for its entire scope (Seg: SCALA) Dynamically typed languages bind the type to the actual value referenced by a variable.

answered Nov 29 '16 at 6:37


user3627021
41 ● 2

3

- In a statically typed language, a variable is associated with a type which is known at compile time, and that type remains unchanged throughout the execution of a program. Equivalently, the variable can only be assigned a value which is an instance of the known/specified type.
- In a dynamically typed language, a variable has no type, and its value during execution can be anything of any shape and form.

answered Oct 29 '16 at 20:18


Chudong
71 ● 5

2

Statically typed languages like C++, Java and Dynamically typed languages like Python differ only in terms of the execution of the type of the variable. **Statically typed** languages have static data type for the variable, here the data type is checked during compiling so debugging is much simpler...whereas **Dynamically typed** languages don't do the same, the data type is checked which executing the program and hence the debugging is bit difficult.

Moreover they have a very small difference and can be related with **strongly typed** and **weakly typed** languages. A strongly typed language doesn't allow you to use one type as another eg. C and C++ ...whereas weakly typed languages allow eg.python

answered Aug 17 '18 at 10:21


Akash sharma
351 ● 2 ● 5

Statically Typed

2

The types are checked before run-time so mistakes can be caught earlier.

Examples = c++

Dynamically Typed

The types are checked during execution.

Examples = Python

edited Oct 5 '19 at 23:00

Tom Zych

11.7k ● 9 ● 31 ● 50

answered Oct 5 '19 at 22:59

Atticus Denewmont

166 ● 1 ● 11

2 This doesn't really add anything that's not yet covered by other answers, does it? – Robert Oct 12 '19 at 17:55

1 Yes, but most answers weren't very clear so I wanted an answer that was easy to understand. – Atticus Denewmont Oct 12 '19 at 17:56

1

Static typed languages (compiler resolves method calls and compile references):

- usually better performance
- faster compile error feedback
- better IDE support
- not suited for working with undefined data formats
- harder to start a development when model is not defined when
- longer compilation time
- in many cases requires to write more code

Dynamic typed languages (decisions taken in running program):

- lower performance
- faster development
- some bugs might be detected only later in run-time
- good for undefined data formats (meta programming)

edited Jul 17 '19 at 5:50

answered Jul 16 '19 at 6:09

MagGGG

11.1k ● 2 ● 19 ● 27

0

dynamically typed language helps to quickly prototype algorithm concepts without the overhead of about thinking what variable types need to be used (which is a necessity in **statically typed languag**e).

answered Mar 14 '18 at 16:34

Pavan Kanajar

1

-15

Static Typing: The languages such as Java and Scala are static typed.

The variables have to be defined and initialized before they are used in a code.

for ex. int x; x = 10;

System.out.println(x);

Dynamic Typing: Perl is an dynamic typed language.

Variables need not be initialized before they are used in code.

y=10; use this variable in the later part of code

answered Jun 27 '13 at 7:19

Prakhyat

689 ● 6 ● 13

2 Well, this answer is not completely right. In both languages, the variables must be intialized before they are used. However, in dynamically typed languages, you may choose to leave out the type where it is used. – darshan Jun 12 '14 at 15:37

It looks like you are misusing the term "variables", you should have said "types" instead. – Emir Dec 25 '14 at 10:15

I'd argue that Perl is statically typed: It has 3 types, scalar (`$`), array (`@`) and hash (`%`). The type of a variable in Perl is known at compile time and stays the same for the rest of the variables lifetime. – CoffeeTableEspresso Jul 4 '19 at 18:50