Day 1

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; other languages (e.g.: Java, C, C++) 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, Erlang

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](http://www.rascal-mpl.org/), takes a hybrid approach allowing dynamic typing within functions but enforcing static typing for the function signature.

2. Scripting languages are programming languages that don't require an explicit compilation step.

For example, in the normal case, you have to compile a C program before you can run it. But in the normal case, you don't have to compile a JavaScript program before you run it. So JavaScript is sometimes called a "scripting" language.

This line is getting more and more blurry since compilation can be so fast with modern hardware and modern compilation techniques. For instance, V8, the JavaScript engine in Google Chrome and used a lot outside of the browser as well, actually compiles the JavaScript code on the fly into machine code, rather than interpreting it. (In fact, V8's an optimizing two-phase compiler.)

Also note that whether a language is a "scripting" language or not can be more about the environment than the language. There's no reason you can't write a C interpreter and use it as a scripting language (and people have). There's also no reason you can't compile JavaScript to machine code and store that in an executable file (and people have). The language Ruby is a good example of this: The original implementation was entirely interpreted (a "scripting" language), but there are now multiple compilers for it.

Some examples of "scripting" languages (e.g., languages that are traditionally used without an explicit compilation step):

* Lua
* JavaScript
* VBScript and VBA
* Perl

And a small smattering of ones *traditionally* used with an explicit compilation step:

* C
* C++
* D
* Java *(but note that Java is compiled to bytecode, which is then interpreted and/or recompiled at runtime)*
* Pascal

...and then you have things like Python that sit in both camps: Python is widely used without a compilation step, but the main implementation (CPython) does that by compiling to bytecode on-the-fly and then running the bytecode in a VM, and it *can* write that bytecode out to files (.pyc, .pyo) for use without recompiling.

That's just a *very* few, if you do some research you can find a lot more.

Programming Paradigms

*A paradigm is a way of****doing****something (like programming), not a concrete thing (like a language). Now, it’s true that if a programming language L happens to make a particular programming paradigm P easy to express, then we often say “L is a P language” (e.g. “Haskell is a functional programming language”) but that does not mean there is any such thing as a “functional language paradigm”.*