Topic 1.3: Static versus Dynamic Typing

Data at the Hardware Level

At the hardware level, a microprocessor understands only one thing: binary data. To make this data useful, the processor interprets it according to specific instructions. The most basic data types supported directly by microprocessor architectures are integers and floating-point numbers. Integers are typically stored in fixed-width registers (e.g., 32-bit or 64-bit) and operated upon by the arithmetic logic unit (ALU). Floating-point numbers follow the IEEE 754 standard, storing values as a combination of sign, exponent, and mantissa, and are manipulated by a dedicated floating-point unit (FPU) or vectorized SIMD instructions for parallel processing. From the processor's perspective, a 64-bit value in memory is just bits—whether it represents an integer, a floating-point number, or part of a larger data structure depends entirely on the instructions used to manipulate it.

Primitive Types

In statically-typed languages, such as Java, C, or Rust, there is a small set of *primitive types* that map directly to hardware-level representations. This type information serves multiple purposes, chief among them enabling efficient code generation. Because the compiler knows exactly how many bytes each variable occupies and what operations are valid, it can generate instructions that operate directly on raw bits in registers without runtime type checks. An integer addition becomes a single CPU instruction, not a sequence of type inspections followed by conditional dispatch. The table below summarizes the primitive types available in Java:

Type	Bits	Range	Hardware Mapping
`byte`	8	-128 to 127	signed integer (ALU)
`short`	16	-32,768 to 32,767	signed integer (ALU)
`int`	32	-2^{31} to $2^{31}-1$	signed integer (ALU)
`long`	64	-2^{63} to $2^{63}-1$	signed integer (ALU)
`char`	16	Unicode Code Point	unsigned integer (ALU)
`float`	32	IEEE 754 single-precision	floating point (FPU)
`double`	64	IEEE 754 double-precision	floating point (FPU)
`boolean`	?*	True / False*	unsigned integer (ALU)

* the `boolean` type can technically take a single bit in memory, but memory is usually byte-addressable, so it likely will be implemented as an unsigned integer and set to a value of either `0` or `1`.

Beyond Primitives: Structured Data

All but the most simple programs require more complex data structures than the available primitive types.

In C, a `struct` groups related data into a contiguous memory block whose layout is known at compile time. The compiler is able to treat the `struct` simply as a number of primitive-type components, which is very efficient, while the programmer can access the data structure in a more abstract way.

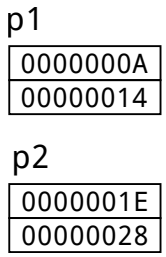
Defining a `struct` in C

```
struct Point {
    int x;
    int y;
};
```

Initializing a `struct` in C

```
struct Point p1;
p1.x = 10;
p1.y = 20;

struct Point p2 = {30, 40};
```

Conceptual diagram of each `struct` in memory



In object oriented languages such as Java and C++, an object similarly groups data fields but adds indirection: variables hold *references* to the memory location where the structure is stored. This indirection enables dynamic allocation and polymorphism. The object data lives in a block of memory called the *heap*, and is accessed via its address.

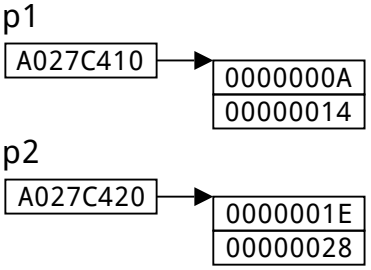
Defining an object in Java

```
public class Point {
   public int x;
   public int y;

   public Point() {}

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
}
```

Initializing an object in Java

```
Point p1 = new Point();
p1.x = 10;
p1.y = 20;

Point p2 = new Point(30, 40);
```

Conceptual diagram of each object in memory



Dynamic Typing in Python: Every Variable is an Object

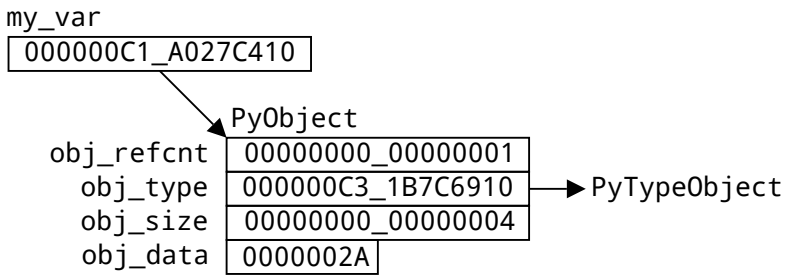
In Python, every piece of data, whether a simple integer, a floating-point number, a string, or a complex object, is represented as an object in memory (on the heap). A variable in Python is simply a name that references one of these heap objects. For example, when Python executes the code:

```
my_var = 42
```

Behind the scenes, Python (here we will discuss the C implementation of Python) creates an integer object on the heap containing:

- A reference count (for garbage collection)
- A type pointer (indicating this is an int)
- A memory structure storing actual value (42);
  for an integer, this contains a size and a number of 32-bit words to store the number

Here is a diagrammatic representation of the object described above, along with the variable `my_var` that stores the reference to this object:



The object type is described by the object `PyTypeObject`, which contains the name of the object (a string), methods that operate on the object, and other essential data about the object. The integer `PyTypeObject` will be shared between all integers that are instantiated. A simple int in Java will occupy 4 bytes (32 bits) of memory. The Python integer object shown above occupies 28 bytes, and if you include the reference pointer, 36 bytes (288 bits) of memory!

This memory structure allows for dynamic typing of variables. Consider the following comparison of Java and Python code that implements the addition of a floating point number with an integer (which have different representations in hardware):

Java Code

```
int x = 1;
double y = 1.0;
double z = x + y;
```

Python Code

```
x = 1
y = 1.0
z = x + y
```

Since the Java compiler knows at compile time the types of `x` and `y` at the time of compile, the bytecode produced will convert the integer, `x`, to a floating point value, then add the two values. When adding two values in Python, the compiler does not know the type of the values it will be adding at runtime. Thus, when the program is running, the Python Virtual Machine will need to check the types of the two values, then call the appropriate method that was written to add those two types (if such a method exists, otherwise raise an exception).

Why Learn C and Java?

Perhaps the greatest benefit of learning C (or Rust) is that it forces the programmer to understand thoroughly the details of how a processor deals with data at a very low level.

Perhaps the greatest benefit of learning Java is that it forces the programmer to thoroughly understand object-oriented data structures and abstractions.

As Python is a very abstract and high-level language, if you only learn to code in Python, you will probably not be forced to understand what is happening “under the hood”. You will be able to happily write inefficient and poorly structured code. If you major in a field such as economics or one of the hard sciences, this is probably sufficient. You just want to write a short, simple program to produce some results, and Python is ideal for this. However, if you expect to write a more capable application, I highly recommend you learn a low-level language such as C or Rust to understand hardware, and learn Java (or perhaps Kotlin) to understand the benefits of object-oriented design. (I do not have experience with functional programming, such as Haskel, but you and I would probably be better coders if we learned that paradigm as well.)

Why Learn Python?

The C language is the more efficient language, while Java can approach the efficiency of C with the advantage of producing compiled bytecode that is highly portable. Python’s dynamic typing comes at a relatively high cost in terms of efficiency... so why learn Python?

Python prioritizes readability and developer productivity above all else, making it the language of choice for rapid development and data science. Its clean syntax and immediate feedback loop make it ideal for beginners and rapid prototyping. But however simple and elegant the language is, the biggest reason to learn Python is the extensive ecosystem.

There are hundreds of thousands of packages for everything from web development to scientific computing. For Artificial Intelligence and Data Science, libraries such as NumPy, Pandas, PyTorch, TensorFlow, and scikit-learn have no serious competitors outside Python, and for these libraries, the heavy lifting is offloaded to compiled C code, enabling near-C performance for numerical workloads. Additionally, when you do encounter problems coding in Python, the language has arguably the largest and most accessible programming community to help.