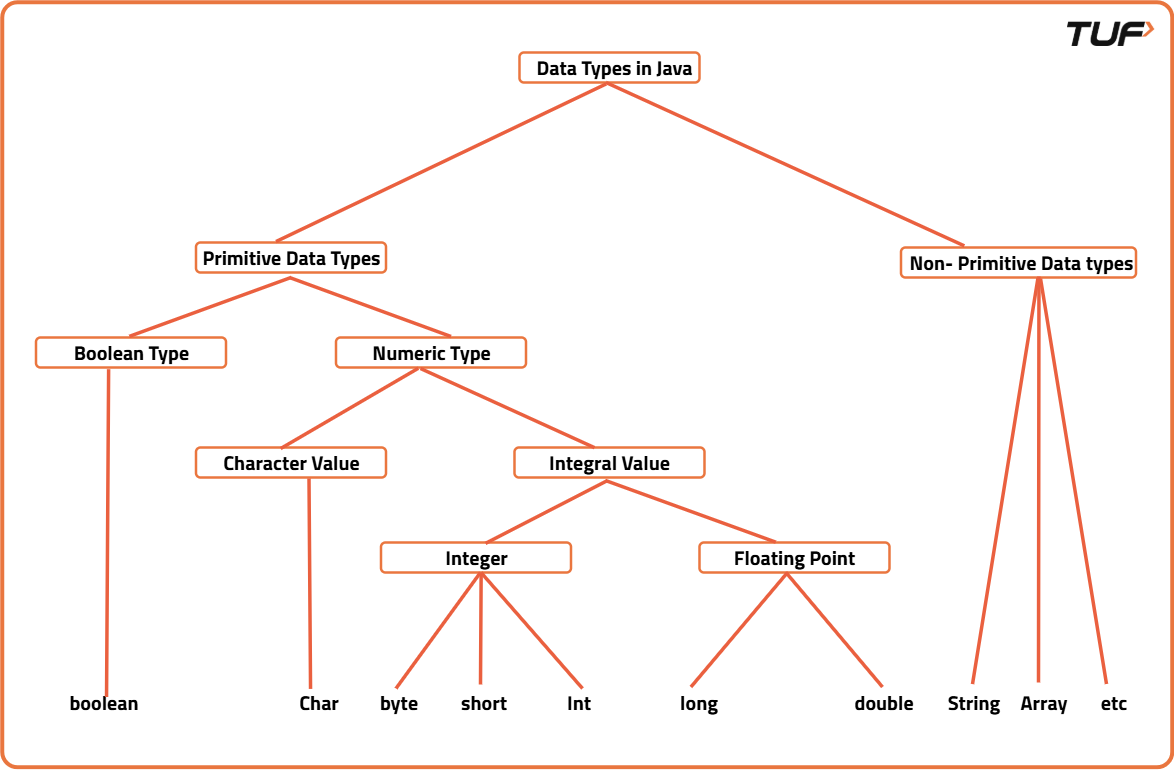
**Data Types in Java**

Java is also statically typed, but unlike C++, it runs on the JVM (Java Virtual Machine) and offers automatic memory management via garbage collection. Data types in Java are split into two broad categories:

* **Primitive Types (fixed in size and stored directly in memory):**
  + byte, short, int, long for integers of increasing size.
  + float, double for floating-point numbers.
  + char for Unicode characters.
  + boolean for logical true/false values.
* **Non-Primitive (Reference) Types:**

These include objects like String, arrays, user-defined classes, and interfaces. A reference type variable holds a memory address pointing to the actual object in the heap.

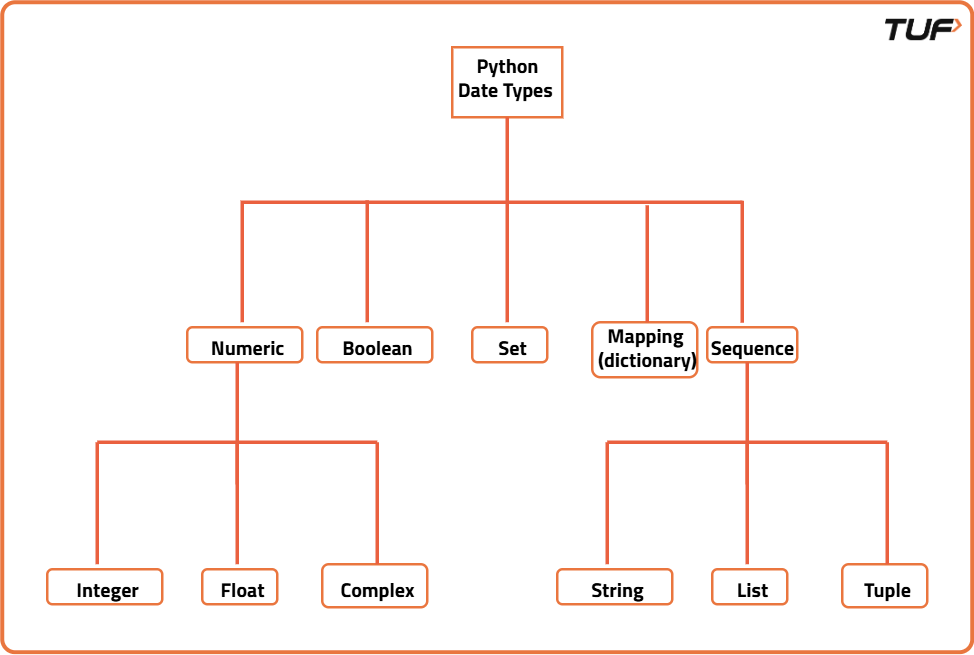


One key difference from C++ is that Java’s boolean type can only be true or false you can’t treat it as a number.

**Data Types in Python**

Python is dynamically typed, meaning you don’t need to declare the data type the interpreter figures it out at runtime. This makes coding faster but can also lead to subtle bugs if you’re not careful.

* Numeric Types:
  + int for integers (unlimited size).
  + float for decimal numbers (double precision).
  + complex for complex numbers (3+5j).
* Sequence Types: list, tuple, and range for ordered collections.
* Text Type: str for strings.
* Mapping Type: dict for key-value pairs.
* Set Types: set and frozenset for unordered collections of unique elements.
* Boolean Type: bool for logical operations.

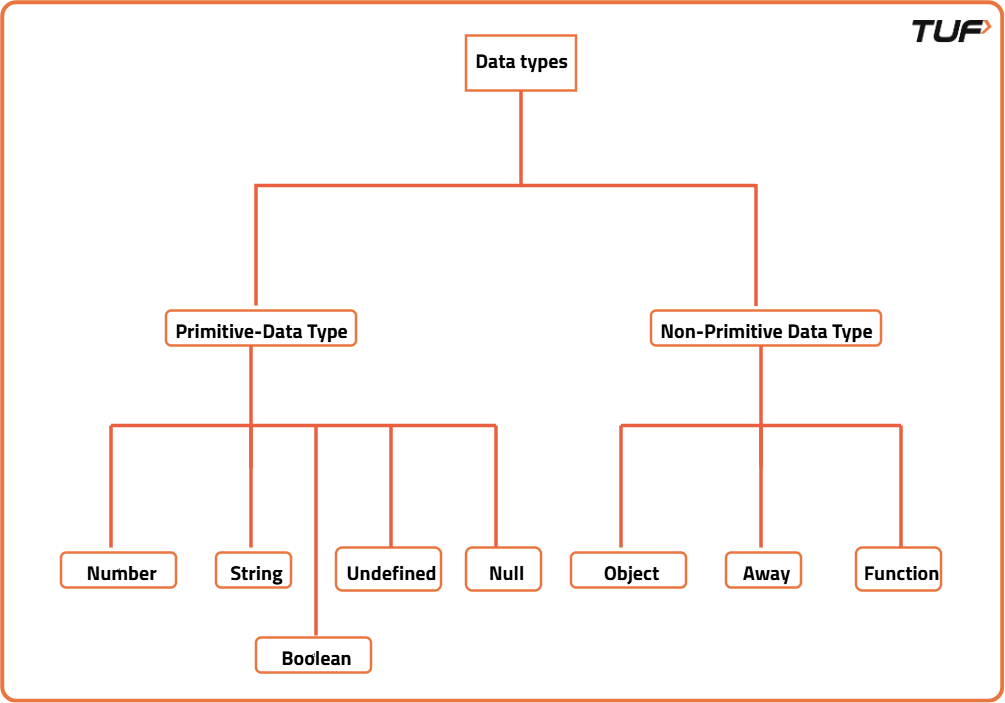


Python is flexible you can assign an integer to a variable, then later assign a string to the same variable without errors. But this flexibility means you have to be more cautious with type-related logic.

**Data Types in JavaScript**

JavaScript is also dynamically typed but is often considered more “loose” than Python when it comes to type conversions. It has fewer built-in types, but its type coercion rules can cause unexpected behavior if you’re not careful.

* Primitive Types:
  + number for both integers and floating-point numbers.
  + number for both integers and floating-point numbers.
  + string for text.
  + boolean for true/false.
  + undefined for variables declared but not assigned a value.
  + null for intentional absence of a value.
  + symbol for unique identifiers.
* Objects: Arrays, functions, and all other non-primitive structures in JS are objects.



One important quirk: In JavaScript, typeof null returns "object", which is a long-standing bug from the early days of the language.

**Key Differences in Practice**

**Memory Management:** Java handles memory automatically with garbage collection, Python uses reference counting with cycle detection, and JavaScript uses mark-and-sweep garbage collection.

**Type Safety:** Java catches type errors at compile time, while Python and JavaScript catch them at runtime, making debugging different across languages.

**Performance:** Java's static typing and JVM optimization often provide better performance, while Python and JavaScript prioritize development speed and flexibility.

| **Category** | **Java** | **Python** | **JavaScript** |
| --- | --- | --- | --- |
| Numeric Types | byte,short,int,long,float,double | int(arbitrary precision),float,complex | number(IEEE-754),bigint |
| Text / Character | char(UTF-16),String | str | string |
| Boolean | boolean | bool | boolean |
| Collections (Core) | ArrayList,LinkedList,HashSet,HashMap,TreeMap, arrays | list,tuple,dict,set,frozenset,range | Array, plain objects{},Map,Set,WeakMap,WeakSet,TypedArray |
| Special / Null | null | None | null,undefined,NaN |
| User-Defined Types | Classes, interfaces, enums, records | Classes (everything is an object), dataclasses, enums, typing hints | Objects, classes (syntax over prototypes), symbols |
| Key Notes | Static typing; primitives vs wrappers (intvsInteger);Stringimmutable | Dynamic typing; immutable vs mutable matters (tuplevslist); integers unbounded | Dynamic typing; loose type coercion;typeof nullreturns"object" |

**Python Array Syntax and Examples**

In Python, there are actually **two main types** of arrays, and they look different:

**1. Lists (Most Common "Arrays")**

Python lists are the most commonly used array-like structure and look like this:

python

*# Basic list syntax*

fruits = ["apple", "banana", "cherry"]

numbers = [1, 2, 3, 4, 5]

mixed = [1, "hello", 3.14, True]

*# Empty list*

empty\_list = []

*# Accessing elements*

**print**(fruits[0]) *# "apple"*

**print**(fruits[-1]) *# "cherry" (negative indexing)*

*# Adding elements*

fruits.append("date")

numbers.extend([6, 7, 8])

**2. Array Module Arrays (True Arrays)**

For homogeneous data types, Python has the array module that creates true arrays:

python

**import** array

*# Integer array syntax*

int\_array = array.array('i', [1, 2, 3, 4])

*# Output: array('i', [1, 2, 3, 4])*

*# Float array*

float\_array = array.array('f', [1.1, 2.2, 3.3])

*# Output: array('f', [1.100000023841858, 2.200000047683716, 3.299999952316284])*

*# Character array*

char\_array = array.array('u', 'hello')

*# Output: array('u', 'hello')*

**Key Differences in Appearance**

**Lists:**

* Use square brackets []
* Can contain mixed data types
* Display as ``

**Array module arrays:**

* Use array.array() constructor
* Must specify type code ('i' for int, 'f' for float, 'u' for unicode)
* Display as array('i', )
* All elements must be the same type

**Common Operations Look Like:**

python

*# List operations*

my\_list = [10, 20, 30]

my\_list[1] = 25 *# [10, 25, 30]*

my\_list.append(40) *# [10, 25, 30, 40]*

length = len(my\_list) *# 4*

*# Array operations*

**import** array

my\_array = array.array('i', [10, 20, 30])

my\_array[1] = 25 *# array('i', [10, 25, 30])*

my\_array.append(40) *# array('i', [10, 25, 30, 40])*

Most Python developers use **lists** for general array-like operations because they're more flexible and easier to work with. The array module is used when you specifically need type-constrained, memory-efficient arrays for numerical computations.