**Data Types: Standard Python Versus Pandas**

In standard Python, each variable data type creates a separate object with its own attributes. For example, when you create an integer:

An object is created with attributes that store information about the variable x, such as its name (x), type (<int>), value (124), and other metadata. This object-oriented approach to handling data types, including integers, can be memory inefficient. For instance, the memory size used by the variable x with the value 124 is relatively large due to the overhead associated with the object attributes.

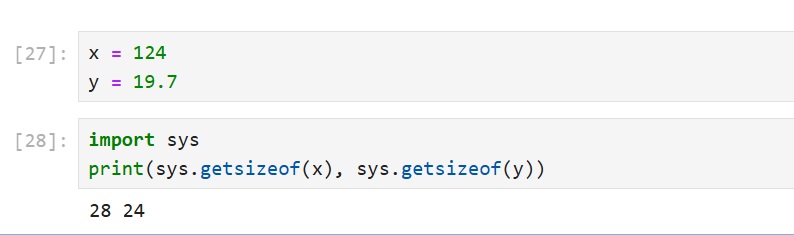


Figure 1: Illustrate memory size of x in Python

Furthermore, Python variables are dynamically typed, meaning their type can change according to the value assigned to them. For example:

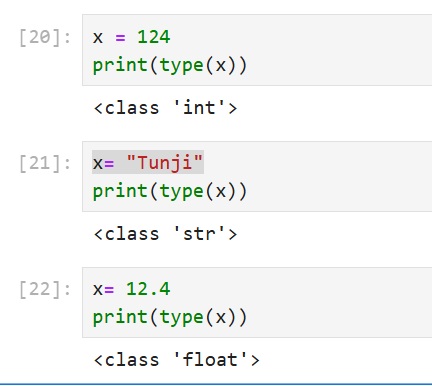


Figure 2: Illustrate dynamic typing in Python

This behaviour contrasts with other programming languages like C, C++, or Java, where data types such as integers are primitive types stored directly in memory. In these languages, an integer or float variable, such as x and y, would typically use a fixed amount of memory—just 4 bytes for an integer in Java (see Figure 3), compared to the larger memory footprint in Python as illustrated in Figure 1.

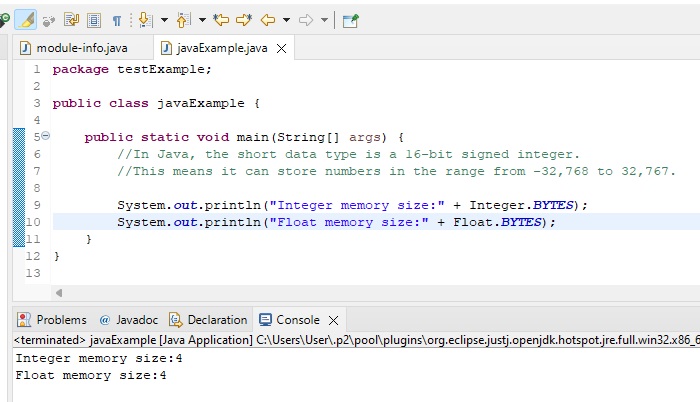


Figure 3: Memory size allocation for Integer and Float data type in Java

In C, C++, and Java, variables are statically typed, requiring you to declare the type of a variable before assigning a value to it. You cannot assign a value of a different type to a variable once its type has been declared. For example:

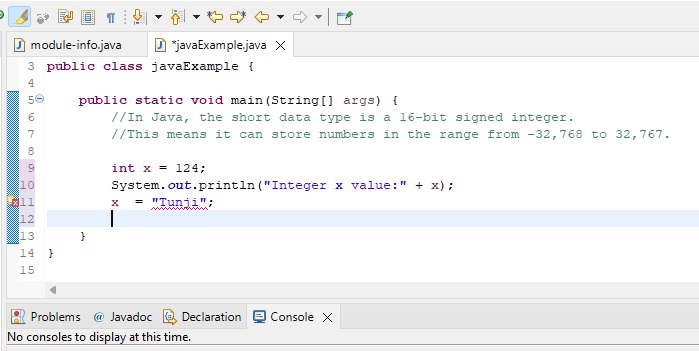


Figure 4: Illustrate type mismatch error

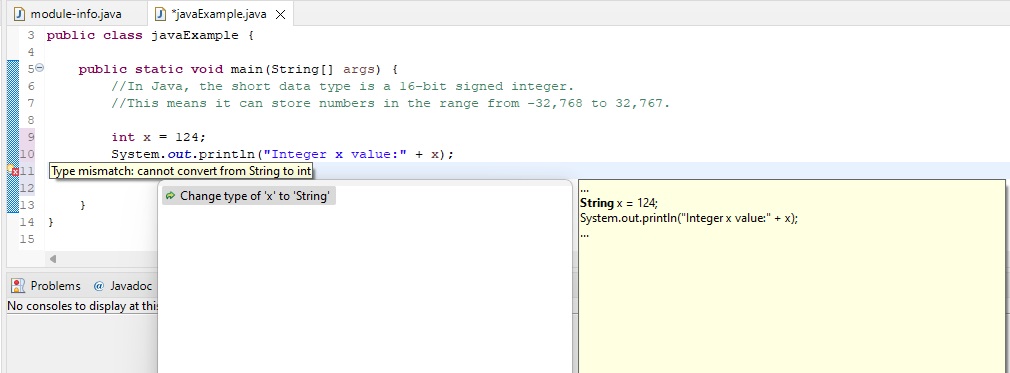


Figure 5: System suggests correction of type mismatch error.

Additionally, for types such as integers or floats, you cannot assign a value beyond the memory size allocated for the type. If you do, the value will be adjusted to fit within the allocated memory. In Java, attempting to assign a value outside the range of an integer type will result in a modified value that fits within the integer's memory allocation.

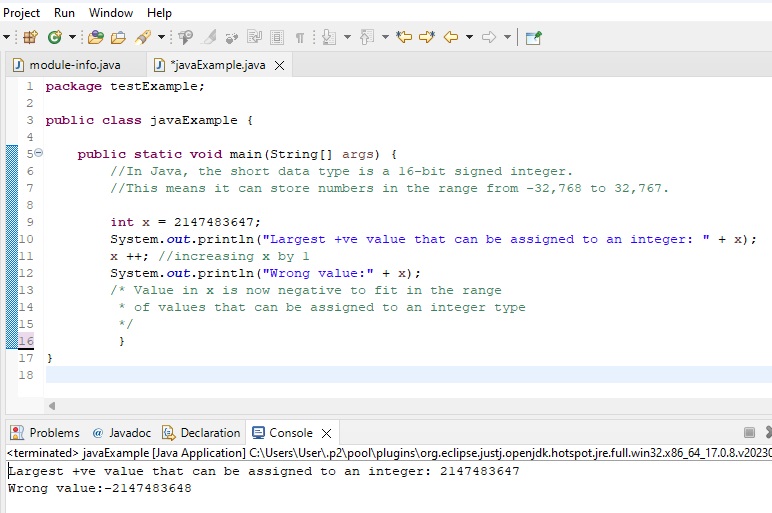


Figure 6: How Java handle value assigned to a variable that that is above the range of allocated memory size.

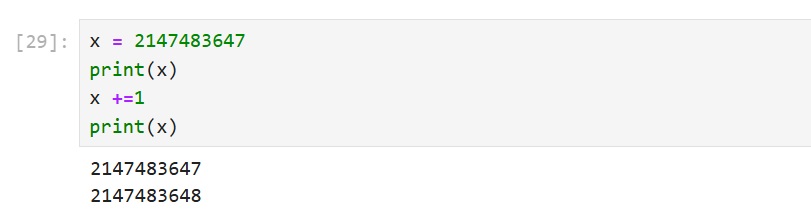


Figure 7: No maximum digit for numeric variable in Python, you can assign as much as you like.

While Python's dynamic typing and ability to handle arbitrarily large integers can be advantageous for single numeric values, it becomes a significant drawback when dealing with large datasets containing thousands or millions of numeric values. The large memory usage due to each numeric value being an object, combined with the dynamic typing allowing for extensive digit allocations, can lead to inefficiencies in Python.

You might wonder why Python is so popular for data science despite its drawbacks in handling numeric data. The answer lies in the Numerical Python (NumPy) library, which is highly memory efficient for managing numeric and homogeneous data. However, NumPy has its own limitation: it is not as efficient with heterogeneous data, and most datasets we want to analyse include columns with various types such as text, numbers, and dates.

This is where the Pandas library comes to the rescue. Pandas offers a more memory-efficient approach for handling large datasets by using data structures optimized for performance and memory usage. Pandas leverages NumPy internally to handle numeric data, allowing it to store homogeneous data in more compact forms. It also provides efficient data manipulation capabilities, thus mitigating some of the memory inefficiencies associated with standard Python data types.