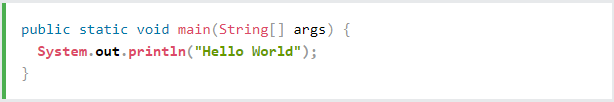
**Boiler Plate Code** are sections of code that are repeated in multiple places with little to no variation

**Verbose:** The programmer must write a lot of code to accomplish only minor functionality. Such code is called *boilerplate*.

Creating a function called “main” … this is a special function & will be executed every time you run a class

**Syntax Explained**



Print Statement: To print something out in the console.

**Note:** The curly braces {} marks the beginning and the end of a block of code.

**Note:** Each code statement must end with a semicolon.

**Comments:**

**//** This is a line comment

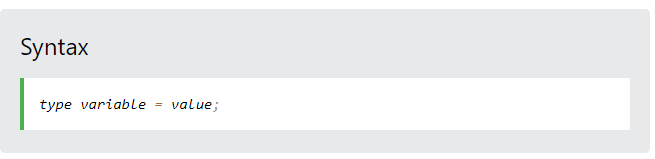
**/\*** Multi-Line

Comment **\*/**

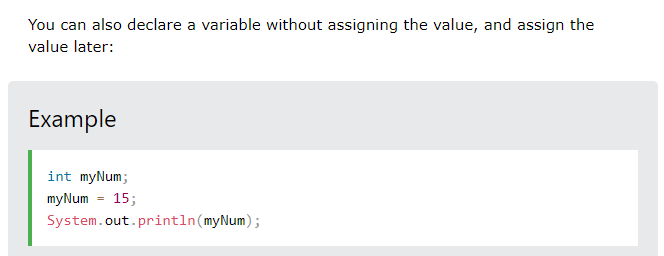
Normally, we use // for short comments, and /\* \*/ for longer.

**Basic Keywords**

**Variable Introduction**



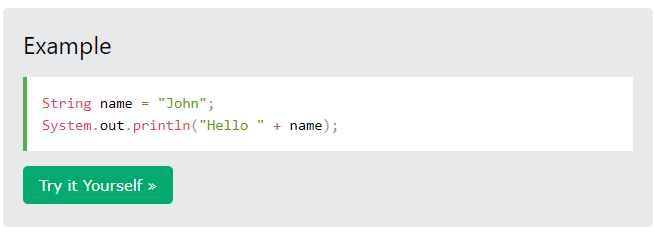
*You can also declare a variable without assigning the value, and assign the value later*



*However, you can add the final keyword if you don't want others (or yourself) to overwrite existing values (this will declare the variable as "final" or "constant", which means unchangeable and read-only):*

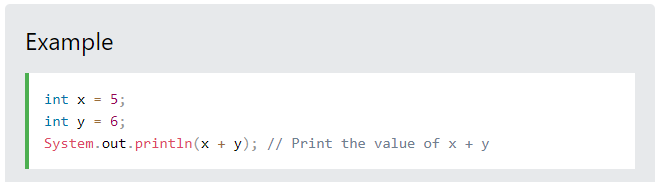
**Note:** If you assign a new value to an existing variable, it will overwrite the previous value:

To combine both text and a variable, use the + character

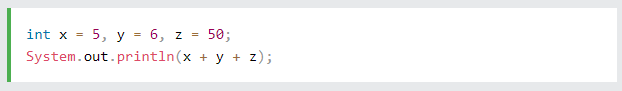




For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):



To declare more than one variable of the **same type**, use a comma-separated list:



**Java Identifiers**

All Java **variables** must be **identified** with **unique names**.

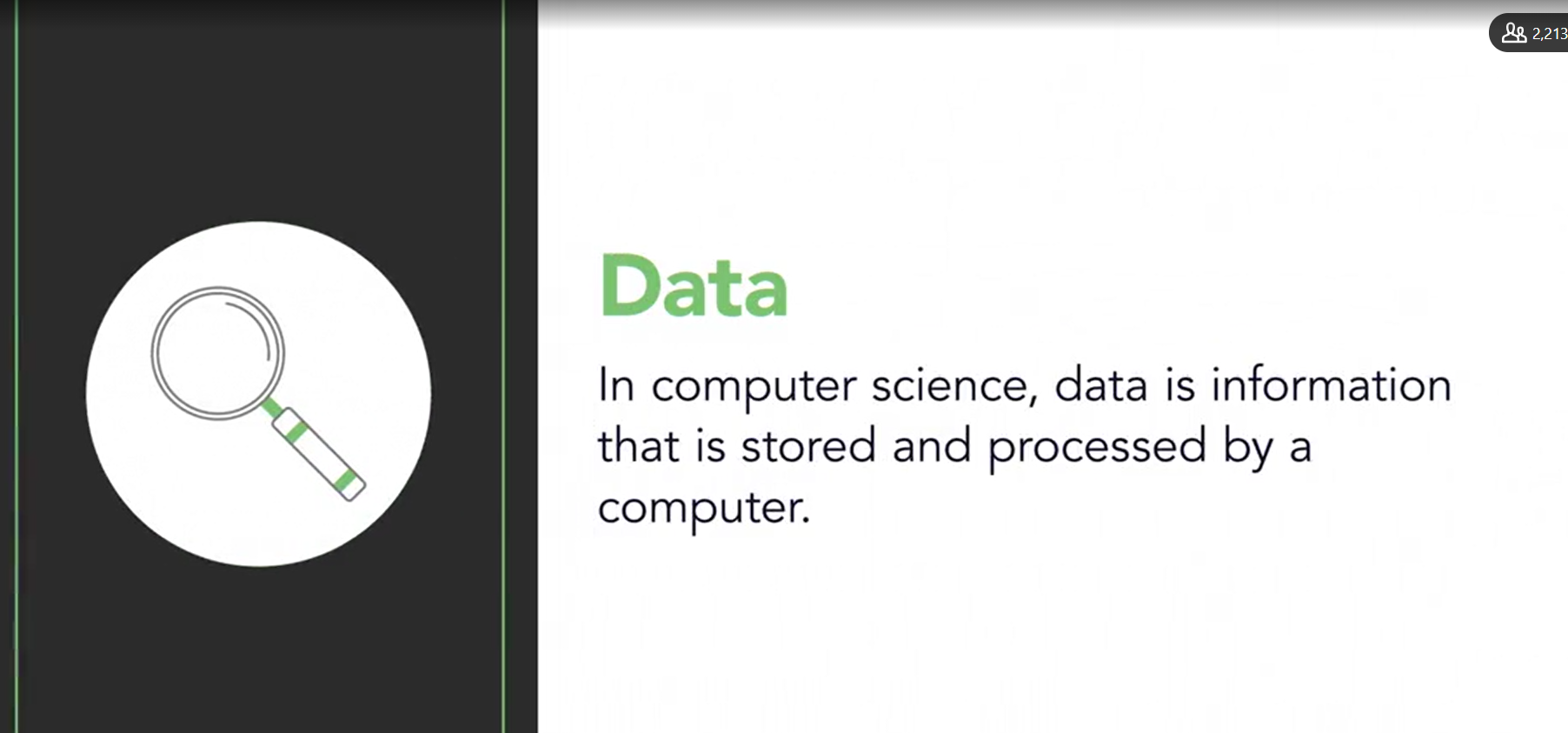
These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

* Names can contain **letters, digits, underscores, and dollar signs**
* Names must begin with a letter
  + Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names should start with a **lowercase letter and it cannot contain whitespace**
* Names are **case sensitive** ("myVar" and "myvar" are different variables)
* **Reserved keywords** (like Java keywords, such as int or boolean) cannot be used as names

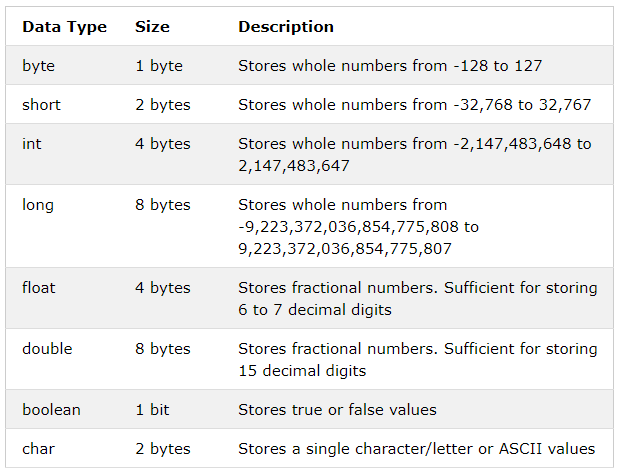
**Java Data Types**



**Primitive Data Type**

A primitive data type specifies the size and type of variable values, and it has no additional methods.

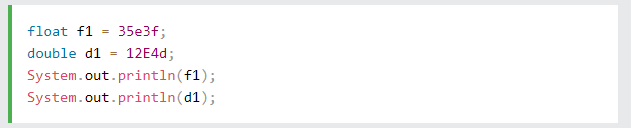
There are eight primitive data types in Java:



Use float or double?

*The***precision***of a floating point value indicates how many digits the value can have after the decimal point. The precision of*float*is only six or seven decimal digits, while*double*variables have a precision of about 15 digits. Therefore it is safer to use*double*for most calculations.*

*A floating-point number can also be a scientific number with an "e" to indicate the power of 10*



**Reference (Non-Primitive) Types**

* **String**: a sequence of ordered characters

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
* The size of a primitive type depends on the data type, while non-primitive types have all the same size.

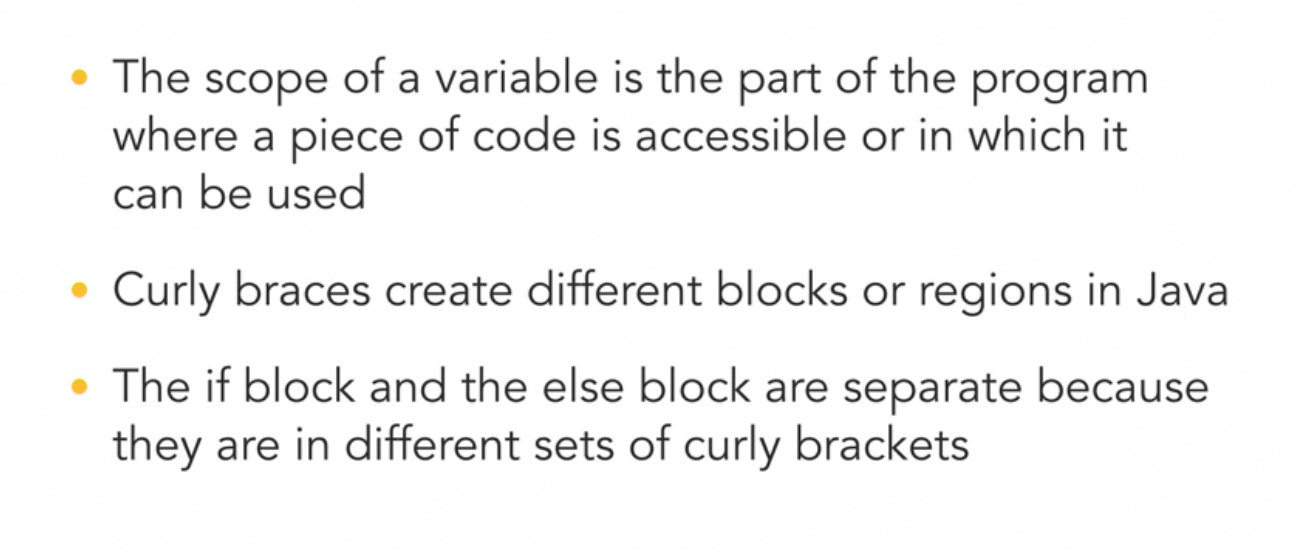
Examples of non-primitive types are [Strings](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp), [Classes,](https://www.w3schools.com/java/java_classes.asp)[Interface](https://www.w3schools.com/java/java_interface.asp), etc. You will learn more about these in a later chapter.

* **Boolean Expression**: Can be used as condition for *decision blocks*
* *Condition blocks* can only result in a Boolean value (true or false)

**Relational Operators:**

* <
  + Lesser Than
* >
  + Greater Than
* ==
  + Equal To
* >=
  + Greater than, equal to
* <=
  + Lesser than, equal to
* !=
  + Not equal to: False if the number is same and true in any other case

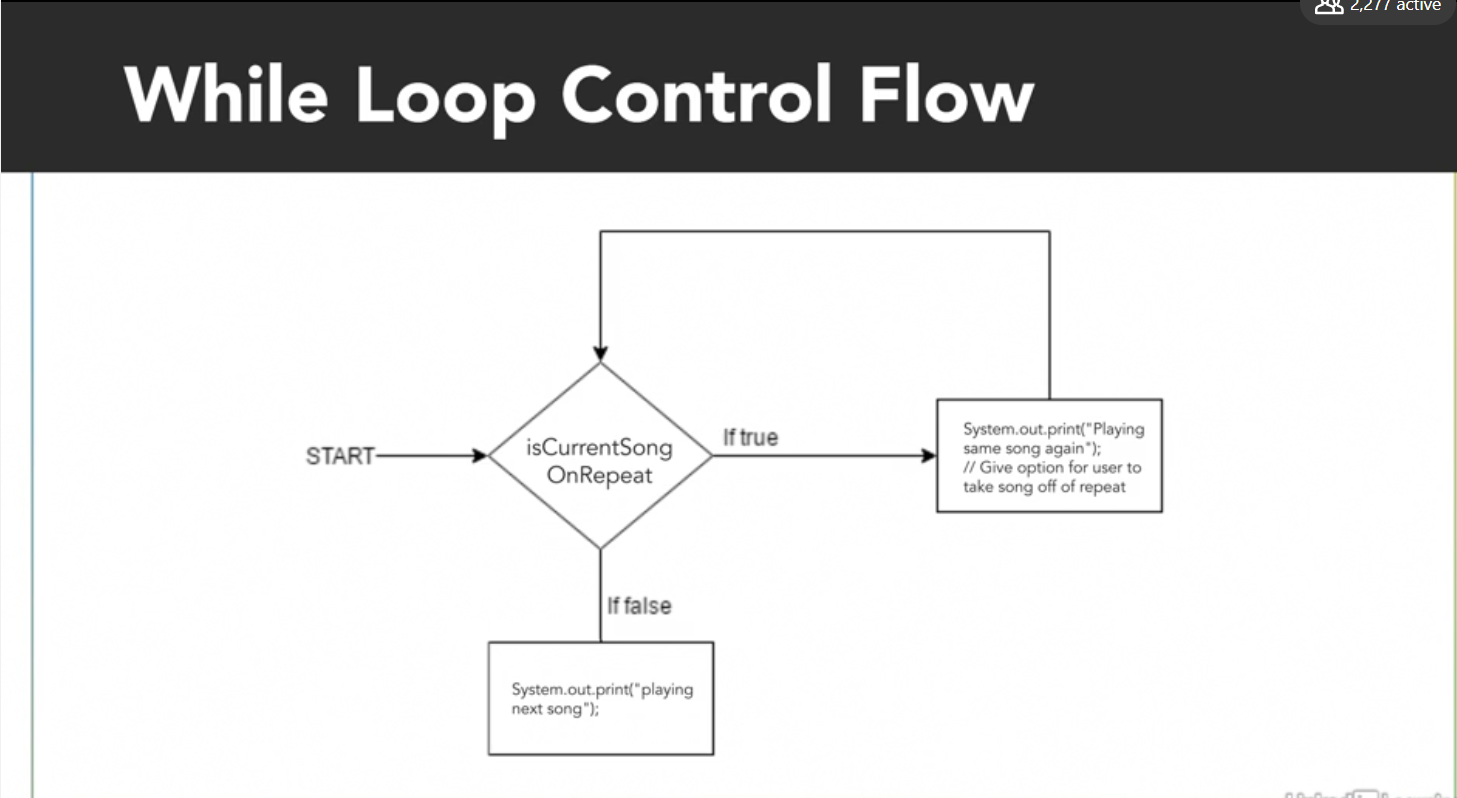
**Scope**



The curly brackets define the scope… any variable defined inside that curly bracket can only be used there or in the child block…

**Loops** :

**While Loop:** Till the the time something is true… keep looping the code inside.



**Input** :

STRING INPUT –

Scanner scanner = new Scanner(System.in);

String input = scanner.next();

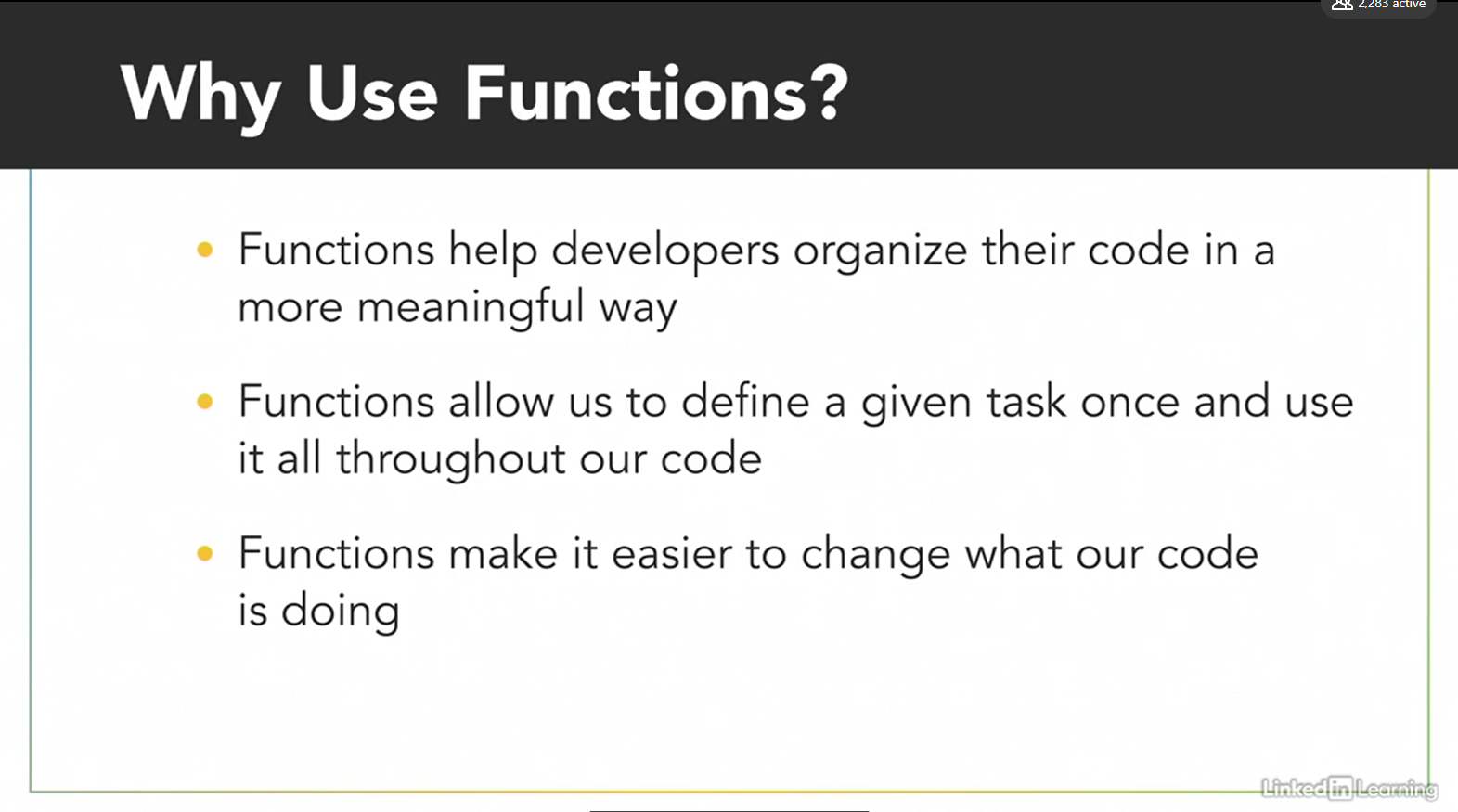
Other INPUT –

Scanner input = new Scanner(System.*in*);  
int input = input.next<var\_type>();

**Errors**

* **Syntax Error** *(spelling, symbol etc… anything related to the syntax or text of the code)*
* **Logical Error**: Program runs but not correctly; related to the logic applied in the program

**Functions**



*If you return a value of a function inside a var, it will execute the function and only save the value…*

**Classes**

* *Classes help us organize our code*
* A class Contains
  + Attributes (properties): **Variables**
  + Behaviour: **Functions**
  + **Constructor**: A function that helps create an instance of the class

The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

**Note:** Java is case-sensitive: "MyClass" and "myclass" has different meaning.

**Instance vs class methods**

**Instance Method/Variable**

If you need to refer an instance (a var with all class values) in order to execute the method (non-static)

**Class Method/Variable**

If you don’t need to refer to an instance, & you can just mention the class (static)

**HADLE ERRORS (DEBIGGING, IO EXCEPTION, TRY & CATCH BLOCK)**