VARIABLES & ARITHMETIC.

Poggie

02.06.2024

// TODO.

Quick recap on variables

Data types

Arithmetic operators

Lab 1

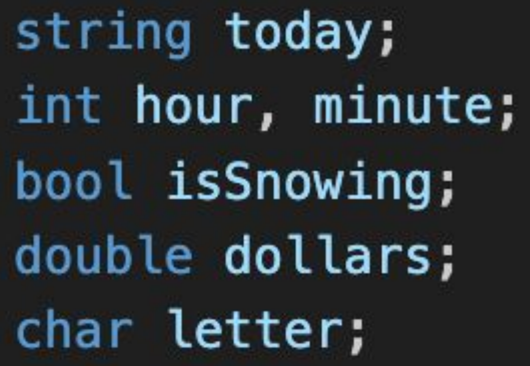
VARIABLES.

Declarations.

A variable is a named location that stores a value.

To store a value we first need to declare a variable.

The following statements declare variables of different types in C#



```
string today;
int hour, minute;
bool isSnowing;
double dollars;
char letter;
```

```
int someNumber;
```

The type of this variable is an int.

It is short for integer.

someNumber is a place in memory with

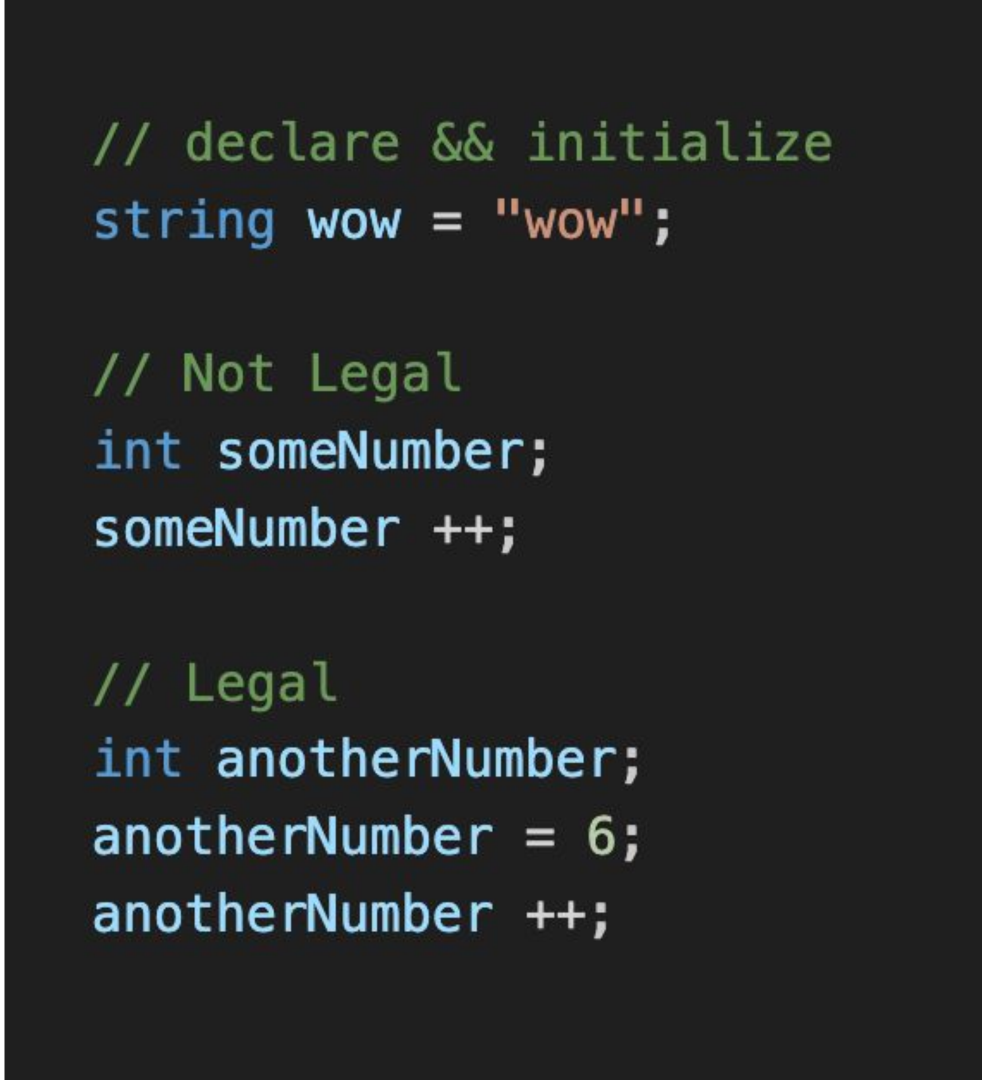
enough space to store an integer.

Assignments.

We can use variables to store values with an assignment statement.

When we make an assignment we update the variable's value.

Variables must be initialized (assigned for the first time) before they can be used.

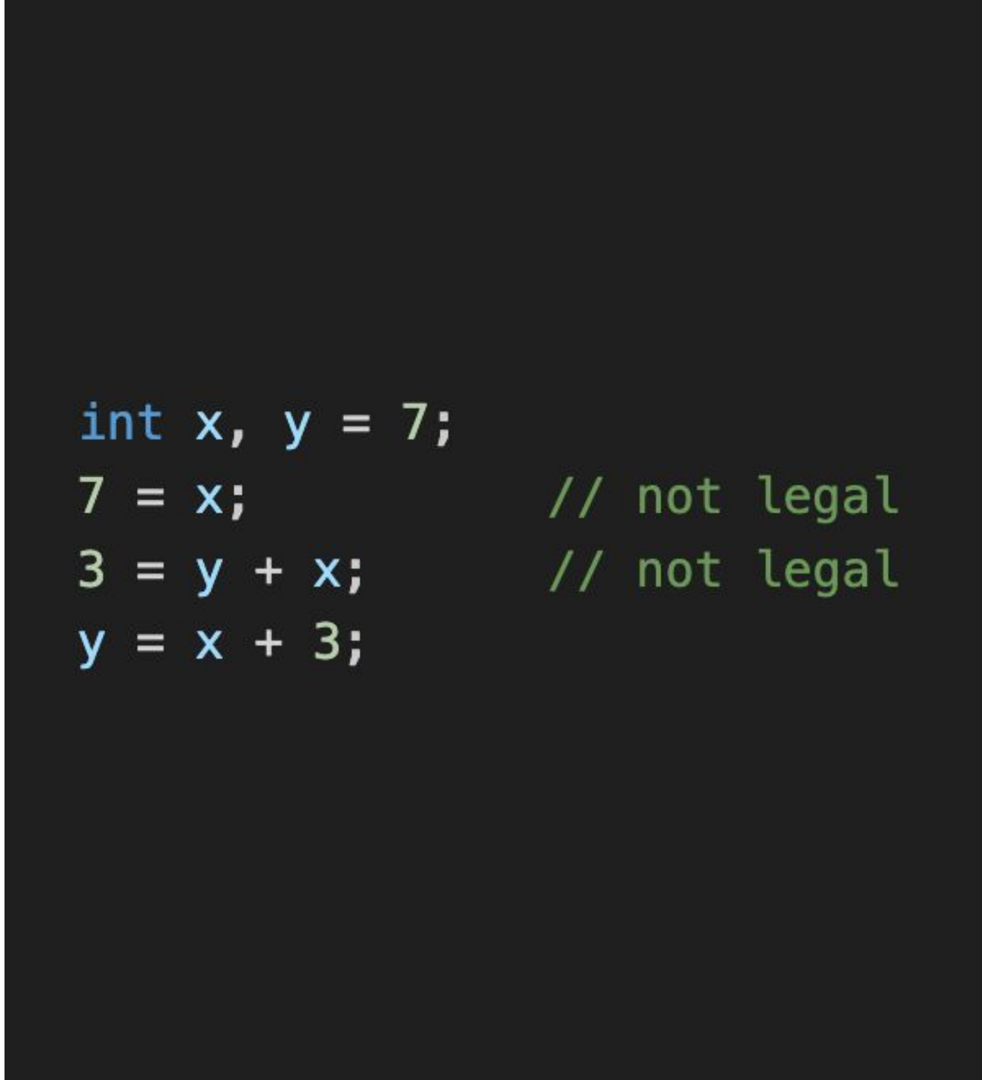


```
// declare && initialize
string wow = "wow";

// Not Legal
int someNumber;
someNumber ++;

// Legal
int anotherNumber;
anotherNumber = 6;
anotherNumber ++;
```

‘Equals’.

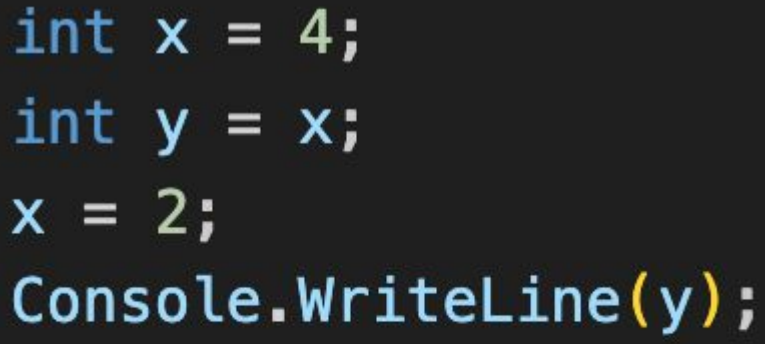
= is an operation assigning the value on the right hand side to the variable on the left hand side.

It is NOT a statement of equality !!

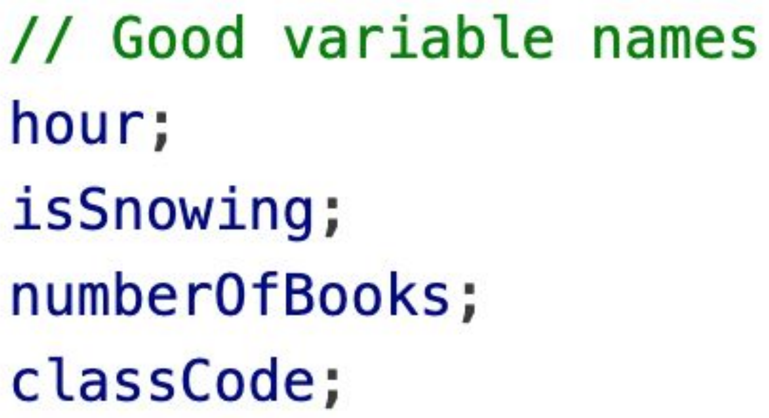


```
int x, y = 7;
7 = x;                  // not legal
3 = y + x;              // not legal
y = x + 3;
```

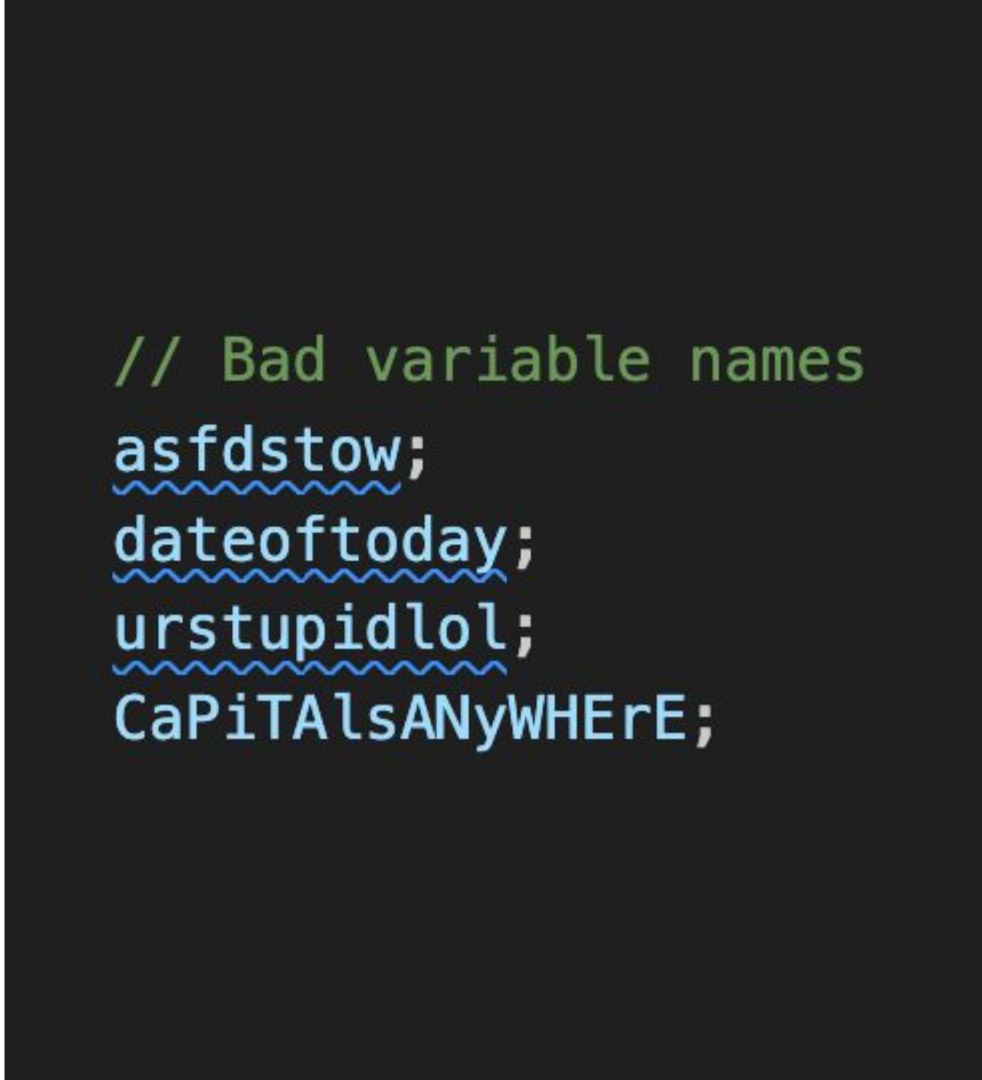
What’s going on here?



```
int x = 4;
int y = x;
x = 2;
Console.WriteLine(y);
```



```
// Good variable names
hour;
isSnowing;
numberOfBooks;
classCode;
```

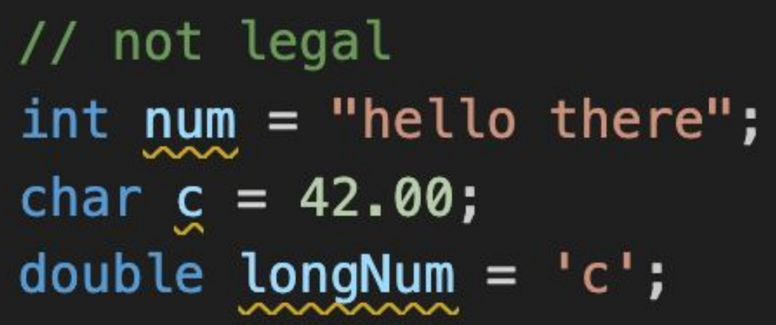


```
// Bad variable names
asfdstow;
dateoftoday;
urstupidlol;
CaPiTAlsANyWHErE;
```

Type checking.

A strongly-typed language enforces strict associations between variables and specific data types.

There will be type errors if variable types do not match up as expected in the expression.



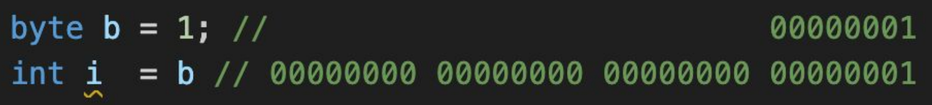
```
// not legal
int num = "hello there";
char c = 42.00;
double longNum = 'c';
```

Implicit type conversion.

Assigning a small size variable to a larger one

No loss of information will occur

Legal, but bad style overall



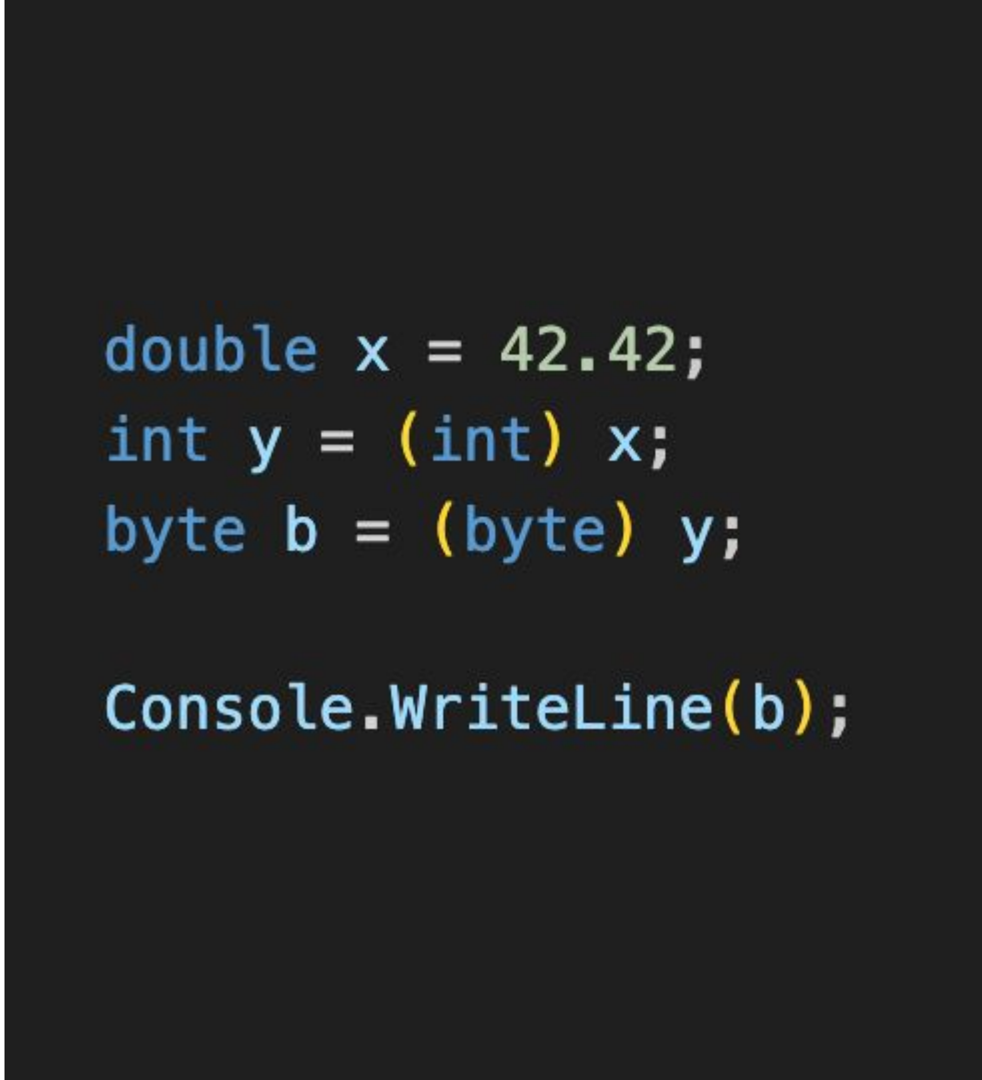
```
byte b = 1; //                               00000001
int i  = b // 00000000 00000000 00000000 00000001
```

Explicit type conversion.

Also known as ‘casting’

Used when store a larger size variable into a smaller one.
(e.g. going from double to int)

Will result in lost information



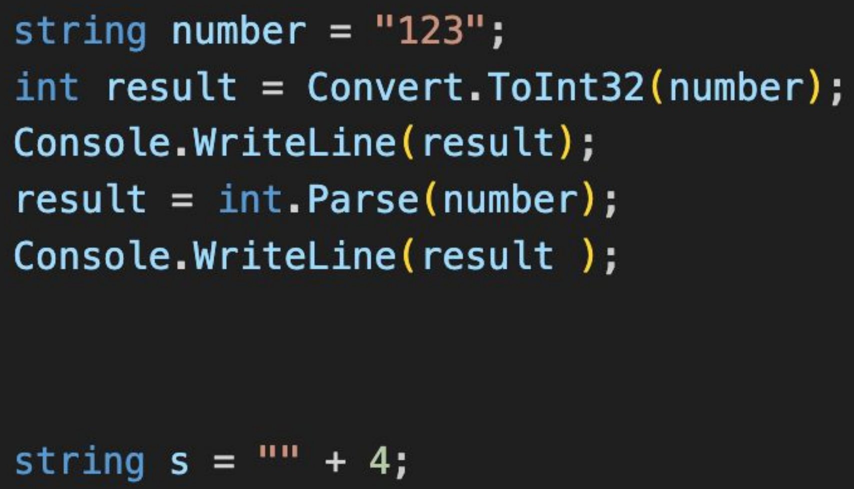
```
double x = 42.42;
int y = (int) x;
byte b = (byte) y;

Console.WriteLine(b);
```

Non-compatible type conversion.

Converting between non-compatible types

Most common scenarios:

- String ⇒ Number
- Number ⇒ String



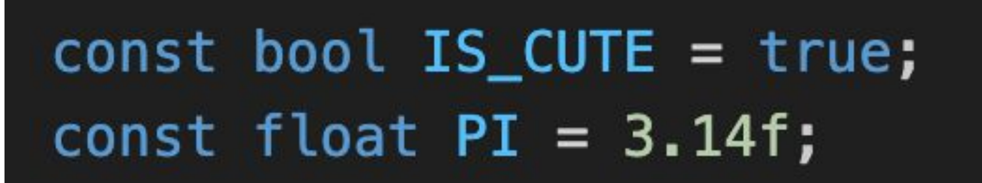
```
string number = "123";
int result = Convert.ToInt32(number);
Console.WriteLine(result);
result = int.Parse(number);
Console.WriteLine(result );



string s = "" + 4;
```

Constants.

Fixed values that the program can not change during execution.

Why?

- Create safety in the application.
- They make widespread changes easier throughout an application.
- Make applications more self-explanatory / easier to follow.

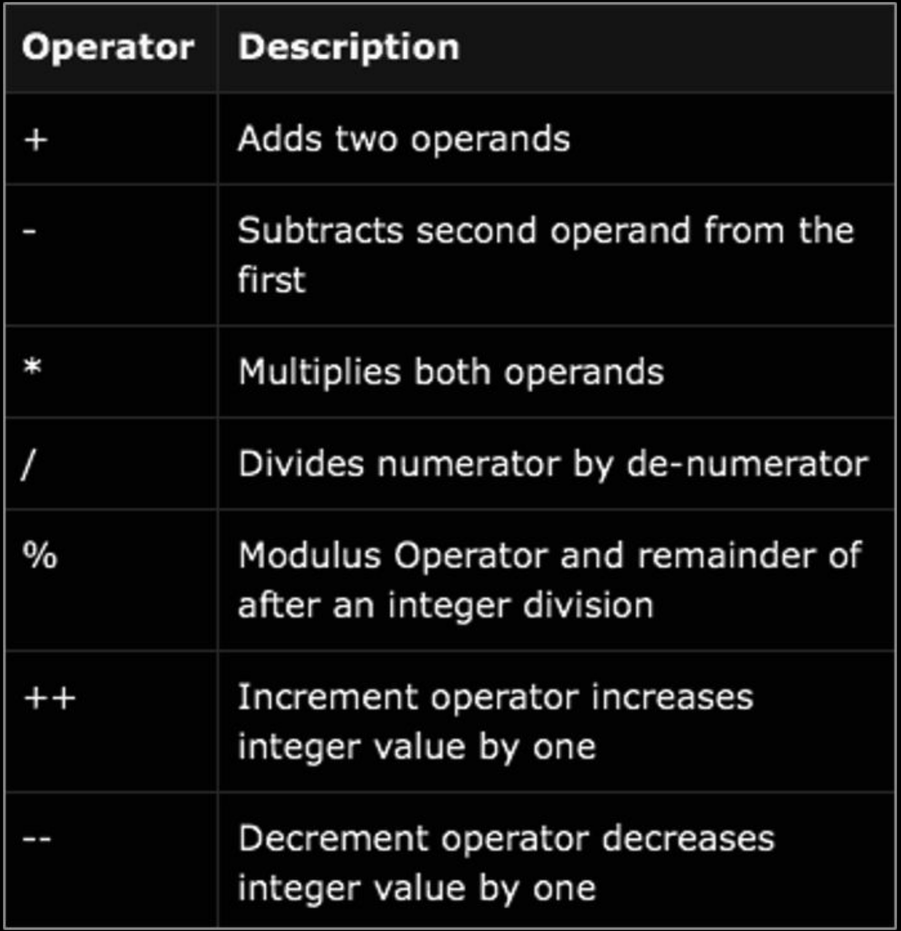


```
const bool IS_CUTE = true;
const float PI = 3.14f;
```

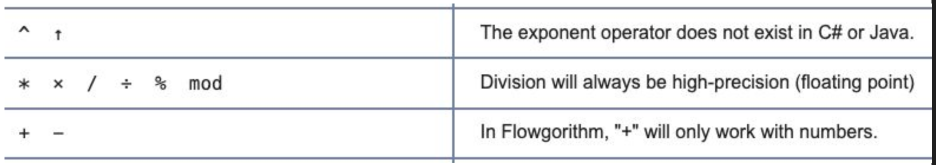
ARITHMETIC OPERATORS.

Supported by C#.

We will be exploring more this thursday !!



Operator	Description
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus Operator and remainder of after an integer division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one



^ ↑	The exponent operator does not exist in C# or Java.
∗ × / ÷ % mod	Division will always be high-precision (floating point)
+ −	In Flowgorithm, "+" will only work with numbers.

Supported by Flowgorithm.

… and more to come on thursday as well :)

LAB TIME.