⊙ Assignment operators: They're used to assign values to a variable or express^n.

i) Simple equals
ii) +=
iii) -=
iv) /=
v) *=
vi) %=

Q. Age = 21. 2040... what'll be person's age

⊙ Declaration vs Initializat^n vs Re-initializat^n
⊙ Concatenation: To use 2 diff. data-types together.
↓ (+)

int age = 21; → 2022

age = → 2040

int age = 21

int age; ✓

age → 21

→ I t was ag ood d ay
→ I tw a s ag o od da y
It ○ was ○ a ○ good
It ○ was ○

Syntax to store data (number):

int x = 20;

data-type | variable | Assignment operator | data | semi-colon

- ⊙ Intro
- ⊙ Syntax
- ⊙ variables, data types and their importance.
- ⊙ Print-statement, semi-colon
- ⊙ Console, compiler
- ⊙ Abstraction.
- ⊙ literal, keywords
- ⊙ comments and naming convention.
- ⊙ Z-Pattern

Operators:

i) Arithmetic operators: +
-
asterisks ← ⊛ → multiply (×)
/ → divide (÷)
IMP * % → modulo operator

This group of operator deals with mathematical expressions

Modulo:

Ten divide by 2:
quotient → 5
2) 10
-10
remainder → 0

3) 12 = 4
-12
0

15 divided by 7
7) 15 = 2
14
1

10 divide by 0

0) 10 = 0.0
0
1
10
...

0 divide 10

10) 0 = 0
0
0

⇒ Compilat^n happens from top to bottom (Except^n)

⊙ Relational operator: They're used to check relation bet^n values.

i) "Equal to" operator (==).
ii) "Not equal to" operator (!=)
iii) "Greater" operator (>)
iv) "Greater equal to" operator (>=)
v) "smaller" and "smaller than equal to".

⊙ Operators indicate the operation we want to perform on operands. Operands indicate the value we want to apply operat^n on.

```
int x = 10;
int y = 20;

boolean z = x <= y;
System.out.println(z);
```

(x, y → operand; <= → operator)