Python basics Notes

✅ What is a Variable?

Ans:-A variable is like a little box where you can keep some information.

📦 It has a name and it holds data.

Example:

name = "Dhruba"

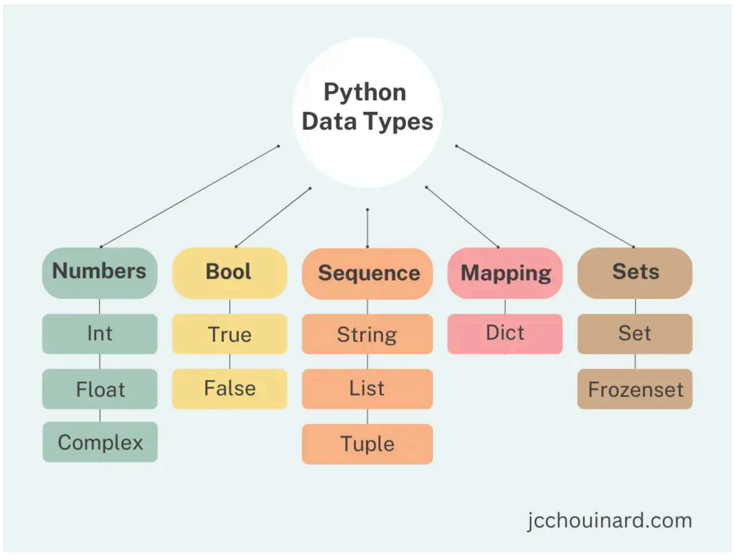
age = 19

temperature = 101.0

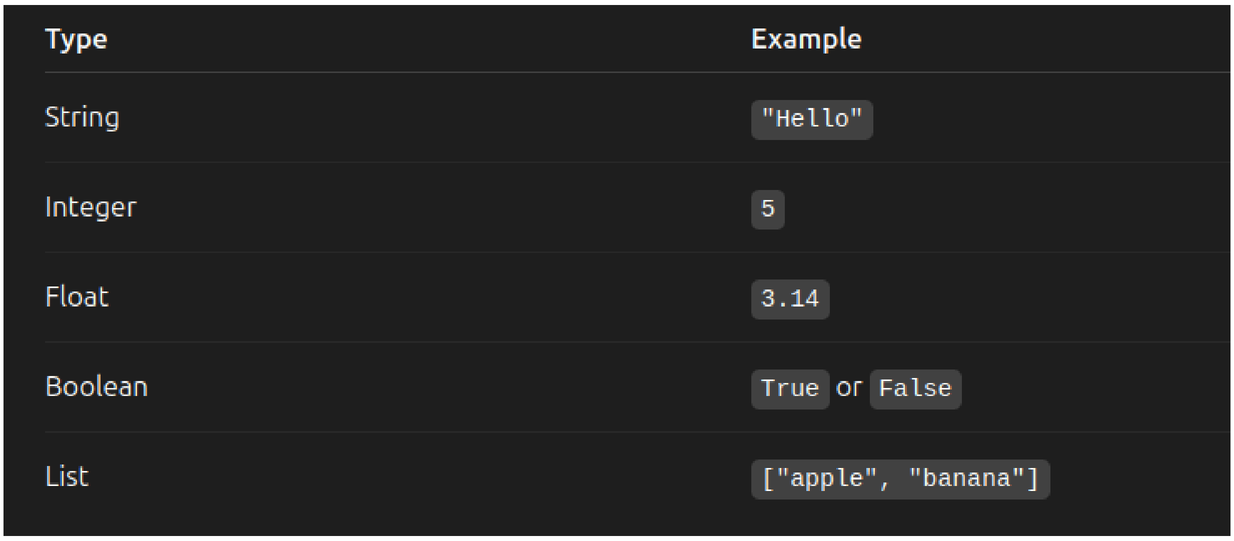
is_sick = True

- `name` keeps text (a string).
- `age` keeps a whole number (integer).
- `temperature` keeps a number with a decimal (float).
- `is_sick` keeps True or False (boolean).

✅ Data Types and its types?



Different kinds of things you can put inside variables:



Type	Example
String	`"Hello"`
Integer	`5`
Float	`3.14`
Boolean	`True` or `False`
List	`["apple", "banana"]`

✅ **Printing**

`print()` shows what's inside your box.

print("Hello, I am learning Python!")

Output- Hello, I am learning Python!

✅ **Type Checking**

`type()` tells you what kind of data is in the variable.

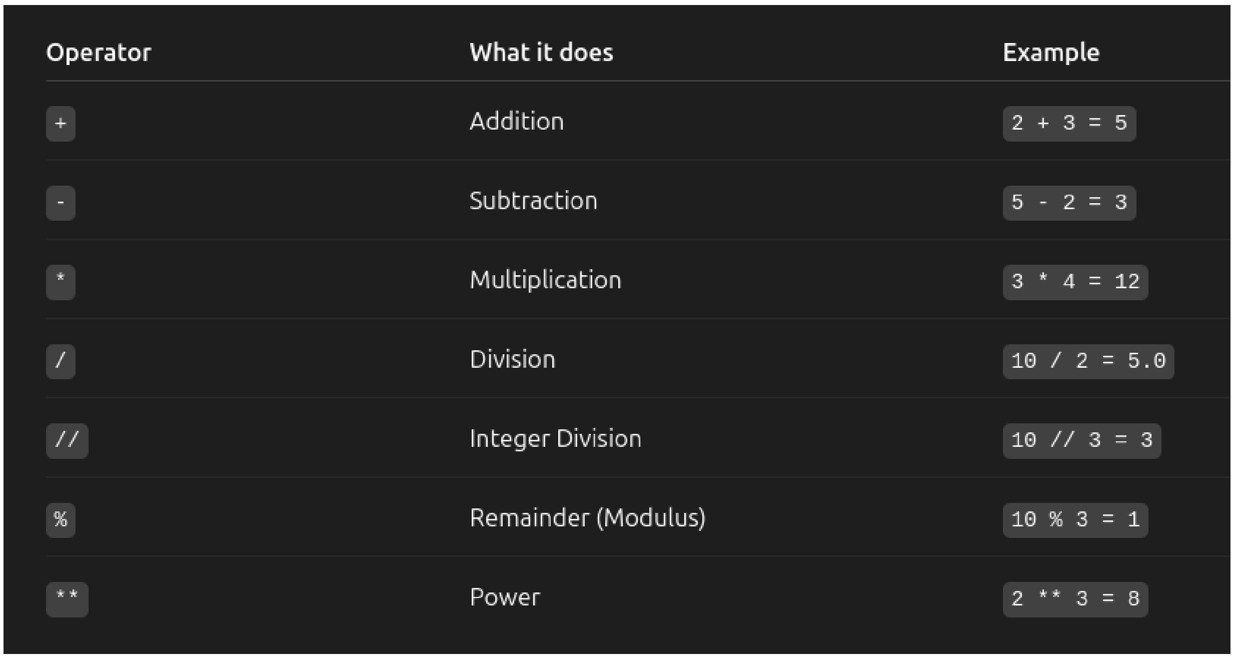
print(type(age))   # Output:<class 'int'>

print(type(name))   # Output:<class 'str'>

✅ Operators and its type?

Operators are symbols that tell Python to do something.

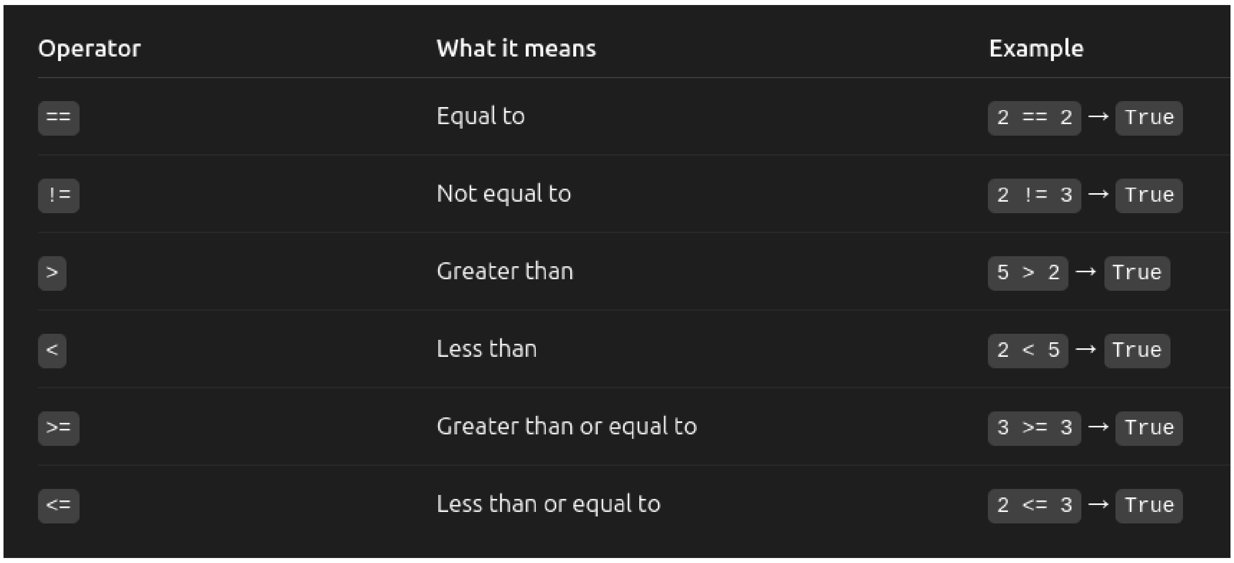
1. Arithmetic Operators:



Operator	What it does	Example
`+`	Addition	`2 + 3 = 5`
`-`	Subtraction	`5 - 2 = 3`
`*`	Multiplication	`3 * 4 = 12`
`/`	Division	`10 / 2 = 5.0`
`//`	Integer Division	`10 // 3 = 3`
`%`	Remainder (Modulus)	`10 % 3 = 1`
`**`	Power	`2 ** 3 = 8`

2.Comparison Operators:

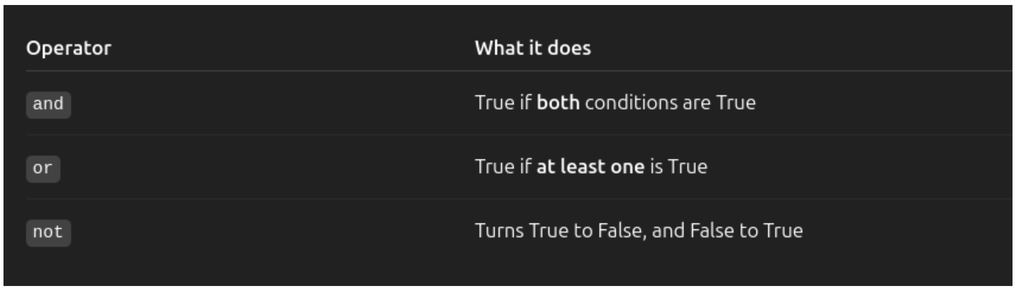
Used to compare values. Always give True or False.



Operator	What it means	Example
`==`	Equal to	`2 == 2` → `True`
`!=`	Not equal to	`2 != 3` → `True`
`>`	Greater than	`5 > 2` → `True`
`<`	Less than	`2 < 5` → `True`
`>=`	Greater than or equal to	`3 >= 3` → `True`
`<=`	Less than or equal to	`2 <= 3` → `True`

3.Logical Operators:

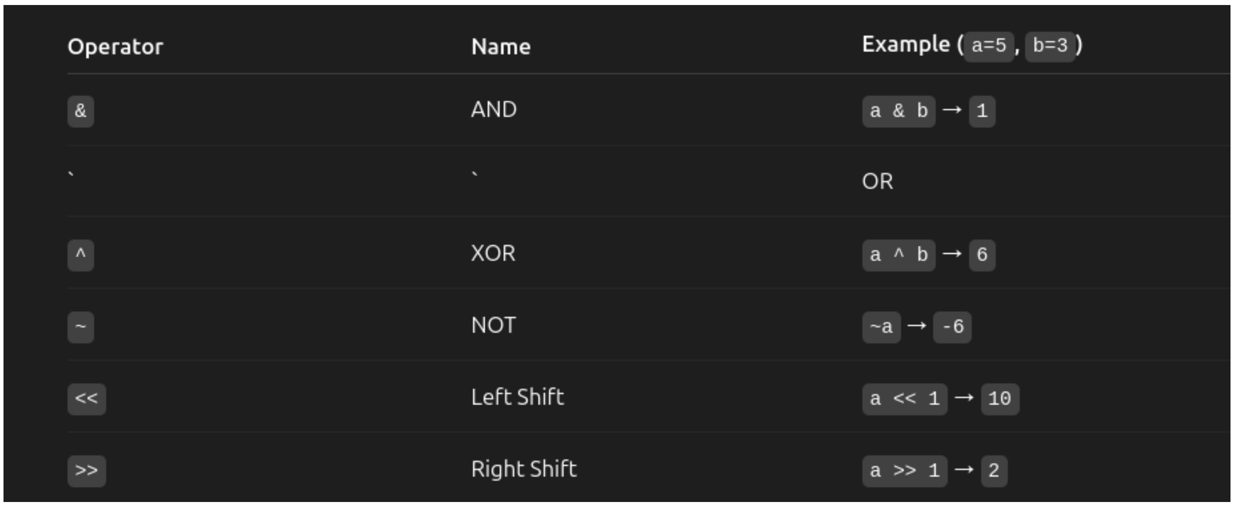
Used to join conditions.



Operator	What it does
`and`	True if **both** conditions are True
`or`	True if **at least one** is True
`not`	Turns True to False, and False to True

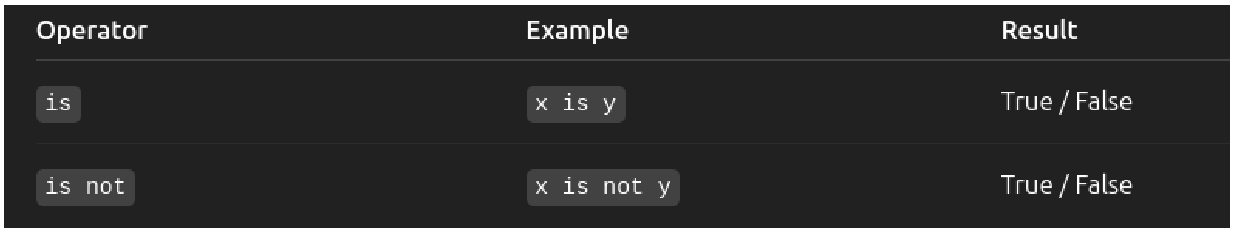
4.Bitwise Operators

Work with bits (0s and 1s of numbers).



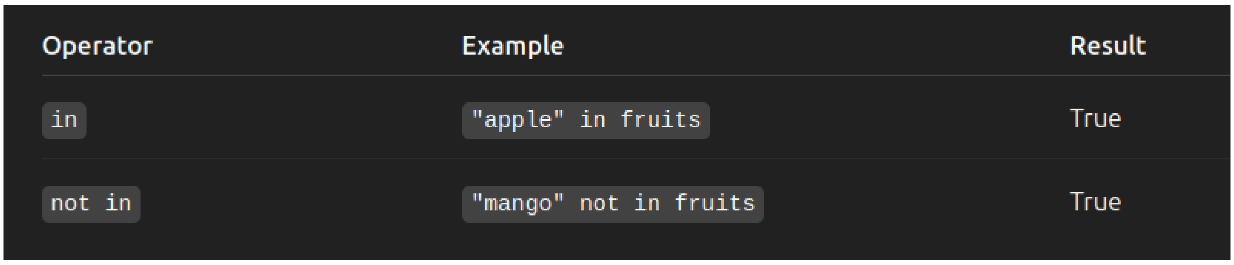
Operator	Name	Example (a=5, b=3)
&	AND	a & b → 1
`	`	OR
^	XOR	a ^ b → 6
~	NOT	~a → -6
<<	Left Shift	a << 1 → 10
>>	Right Shift	a >> 1 → 2

5.Identity Operators
Check if two variables point to the **same object** in memory.



Operator	Example	Result
is	x is y	True / False
is not	x is not y	True / False

6.Membership Operators
See if a value is **inside** something.



Operator	Example	Result
in	"apple" in fruits	True
not in	"mango" not in fruits	True

✅ **Conditional Statements**

Help you **make decisions**.

temperature = 101

```
if temperature > 100:
    print("You have fever.")
elif temperature == 100:
    print("You are on the borderline.")
else:
    print("You are okay.")
```

✅ Loops

Loops help you **repeat things**.

Loops in Python, and in programming generally, are control flow statements that allow a block of code to be executed repeatedly until a specific condition is met or for a defined number of iterations. They are used to automate repetitive tasks, iterate over collections of data, and manage program flow efficiently.

Python primarily offers two types of loops:

- `for` loop:
  - Used for iterating over a sequence (like a list, tuple, string, or range) or other iterable objects.
  - It executes a block of code for each item in the sequence.
  - Example:

  ```
  fruits = ["apple", "banana", "cherry"]
  for fruit in fruits:
      print(fruit)
  ```

- `while` loop:

Executes a block of code as long as a specified condition remains true.

Example:

```
count = 0
while count < 5:
    print(count)
    count += 1
```

Additionally, Python provides loop control statements to alter the normal execution flow of loops:

- `Break`: Terminates the loop entirely and transfers control to the statement immediately following the loop.
- `Continue`: Skips the current iteration of the loop and proceeds to the next iteration.
- `Pass`: A null operation; it does nothing and is used as a placeholder where a statement is syntactically required but no action is desired.

The condition is checked at the beginning of each iteration.

It is crucial to ensure the condition eventually becomes false to avoid an infinite loop.

Input and Output

Input lets the user type something.

```
name = input("Enter your name: ")

print("Hello,", name)
```