Web3

🧠 1. What is Motoko?

- **Motoko** is a statically typed, actor-based programming language.
- Specifically designed for developing smart contracts (called **canisters**) on the **Internet Computer (IC)** by DFINITY.
- It is similar to functional languages like **Haskell**, **OCaml**, or **TypeScript** in syntax but tailored for **web3 and decentralized applications**.

🔷 2. Motoko Basics

🧾 File Extension:

- `.mo`

🧰 Structure of a Simple Canister:

```
motoko
CopyEdit
import Debug "mo:base/Debug";

actor Hello {
  public func greet(name : Text) : async Text {
    return "Hello, " # name;
  };
};
```

🔡 3. Data Types in Motoko

🔢 Numbers:

Type	Description	Example
`Nat`	Natural number (≥ 0)	`100`
`Nat8`, `Nat16`, `Nat32`, `Nat64`	Sized natural numbers	
`Int`	Signed integers (..., -2, -1, 0...)	`-5`
`Int8`, `Int16`, `Int32`, `Int64`	Sized signed integers	

`Float`	Floating point number	`3.14`

🧵 Text and Characters:

Type	Description	Example
`Text`	Unicode string	`"Hello"`
`Char`	Single character	`'A'`

🔘 Boolean:

Type	Description	Example
`Bool`	`true` or `false`	`true`

🧊 Optional:

Type	Description	Example
`?T`	Option type, represents `null`	`?42` , `null`
`null`	Represents absence of value	—

🧱 Compound Types:

Type	Description	Example
`Array<T>`	Fixed-size array	`[1, 2, 3]`
`Buffer<T>`	Growable array	`Buffer.Buffer<Int>(0)`
`Tuple`	Group of values	`(1, "a", true)`
`Record`	Struct-like data	`{ name = "Shivansh"; age = 18 }`
`Variant`	Like enums with values	`#ok(100)` , `#err("Failed")`

🪪 Other:

Type	Description
`Principal`	A user's identity on Internet Computer
`Blob`	Raw binary data
`Canister`	Reference to another actor
`async T`	Asynchronous return type
`actor`	Declares a canister

⚙️ 4. Variable Types

Keyword	Meaning	Example
`let`	Immutable binding	`let x = 5;`
`var`	Mutable variable	`var balance = 100;`
`:=`	Reassignment	`balance := balance + 10;`

🏗️ 5. Functions

🔶 Basic Function:

```
motoko
CopyEdit
func add(x : Nat, y : Nat) : Nat {
  return x + y;
}
```

🔶 Public Function:

```
motoko
CopyEdit
public func greet(name : Text) : async Text {
  return "Hello " # name;
}
```

🔁 6. Control Flow

🔶 If/Else

```
motoko
CopyEdit
if (x > 10) {
  ...
} else {
  ...
```

```
}
```

🔶 Switch (like match)

```
motoko
CopyEdit
switch(optValue) {
  case (?val) { ... }
  case null { ... }
}
```

📦 7. Importing Modules

Use modules from `mo:base`:

```
motoko
CopyEdit
import Nat "mo:base/Nat";
import Debug "mo:base/Debug";
```

Common modules:

- `Nat`, `Int`, `Float`, `Text`
- `Debug`, `Array`, `Buffer`, `HashMap`

🖨️ 8. Debugging Tools

Function	Description
`Debug.print`	Prints to terminal (in dev environment)
`debug_show(x)`	Converts values to string for printing

🛰️ 9. Actor Model

🔷 What is an Actor?

- An **actor** is a **canister** (smart contract) that encapsulates:

- **State** (variables)
- **Public methods**
- **Concurrency** model (each actor processes messages one at a time)

```
motoko
CopyEdit
actor MyCanister {
  var count = 0;

  public func increment() {
    count += 1;
  };
};
```

🧪 10. Asynchronous Functions

- All `public func`s are **async** by default in Motoko.
- Return type is `async T`

```
motoko
CopyEdit
public func getBalance() : async Nat {
  return balance;
}
```

🗃️ 11. Persistent State

Motoko stores state **in memory**, not persistently, unless you:

- Use `stable var`
- Implement `preupgrade` / `postupgrade` hooks

Example:

```
motoko
CopyEdit
```

```
stable var count : Nat = 0;
```

✅ Example: Your Code Explained

```
motoko
CopyEdit
import Int "mo:base/Int";
import Debug "mo:base/Debug";

actor DBank {
  var currentValue = 1600;
  currentValue := 2000;
```

- `DBank` is an actor (smart contract).
- `currentValue` is a mutable `Nat`, reassigned to 2000.

```
motoko
CopyEdit
  public func topUp(amount : Nat) {
    currentValue += amount;
    Debug.print(debug_show (currentValue));
  };
```

- Adds the `amount` to the balance.
- Logs the updated balance.

```
motoko
CopyEdit
  public func withDrawl(amount : Nat) {
    let temp : Int = currentValue - amount;
    if (temp >= 0) {
      currentValue -= amount;
      Debug.print(debug_show (currentValue));
    } else {
      Debug.print("Amount large,currenvalue  <0");
```

```
    };
  };
};
```

- Converts subtraction to `Int` to prevent underflow.
- If sufficient balance: deducts the amount and logs.
- Otherwise: logs warning.

`query` vs `update` in Motoko

On the Internet Computer, there are two main types of public functions you can expose from a canister:

Type	Used For	Modifies State?	Speed	Cost	Requires Consensus?
`query`	Reading only	❌ No	⚡ Very Fast	💰 Free	❌ No
`update`	Reading/Writing	✅ Yes	🐢 Slower	✅ Charged	✅ Yes

🔶 `update` Functions

- **Modifies state**
- Requires **network consensus**
- **Slower**, but changes are permanent and consistent
- Example:

```
motoko
CopyEdit
public func topUp(amount: Nat) {
  currentValue += amount;
}
```

🔶 `query` Functions

- **Read-only** access to state
- No consensus needed → served directly from replica memory

- **Cannot call other** `update` **functions** or change variables
- Used for **reading data only**

Example from Your Code:

```
motoko
CopyEdit
public query func checkBalance() : async Nat {
  return currentValue;
};
```

🧠 What is Orthogonal Persistence?

🔷 Simple Definition:

Orthogonal Persistence means:

> Your program's variables automatically survive canister upgrades without needing to manually save or load them.

🧰 In Practice (Motoko/IC):

- On most platforms, when you restart or upgrade a program, you lose all data in memory unless you save it manually (like writing to a file or DB).
- On the **Internet Computer**, if a variable is declared as `stable`, the **Internet Computer takes care of saving it during upgrades**.
- That's why it's called "**orthogonal**" — it's handled automatically by the platform, not tangled in your business logic.

✅ Example in Motoko:

```
motoko
CopyEdit
stable var balance : Nat = 0;
```

- Declared as `stable` → means it will **automatically be saved and restored** during canister upgrade.

- You **don't need to write** extra code to persist or retrieve it.

⚠️ Without `stable`:

```
motoko
CopyEdit
var balance : Nat = 0;
```

- This will be **lost** when the canister is upgraded or restarted.
- It lives only in **memory** during runtime.

💾 What is `stable` in Motoko?

✅ `stable` Keyword:

> Declares variables that are automatically stored in stable memory, which survives canister upgrades.

🔶 Syntax:

```
motoko
CopyEdit
stable var yourData : Type = initialValue;
```

🌊 Example With Explanation:

```
motoko
CopyEdit
stable var currentValue : Nat = 1000;
```

- `currentValue` is stored in **stable memory**
- When you upgrade the canister:
  - Old version stops
  - New version starts

- Internet Computer **automatically restores** `currentValue`

Arithmetic opr should have same data types