→ <u>Inversion of Control</u> :-
The control of the our 'callback', we are giving to the other function. Now the other function is deciding went + when to call our callback, where to call our callback, whether to call callback or not, how many times to call our callback.

⇒ '.then()' also returns a promise object.
⟶ returns a new promise object

* <u>First Class functions / First Class citizens.</u>
They are called first ~~class~~ functions or first class citizens because they have the ability to be stored in the variables, passed as parameters and arguments. They can also be returned in the function.
- can be used as an argument
- can be executed inside a closure function
- can be taken as return form.

<u>Example</u> :-

```
var b = function(param){
        return function xyz (){
                console.log("F.C.F");
        }
};
```

```
function process(){
    let i = 0;
    function innerProcess(){
        i += 1;
        return i;
    }
    return innerProcess; // we are not calling the
}                          function, we are just returning

let res = process(); // this line calls the function,
                        which returns another function
```

* **Closure :—**

It remembers all those variables that are getting accessed inside our function whose scope might be in our function or outside of the function, it remembers the memory location of all those variables.

Closures makes persistent storage. A editable memory location or memory that gets attached to a function, that it never forgets.

⇒ It is called as 'closures' because we are technically 'closing over values' means it is remembering the scope of a variable, we are not snap-shoting the value means it is not a never changing property. It is actually same variable.

→ Closure :—
"A function binds together with its lexical environment".

Function along with its lexical scope forms a closure.

A closure is a function that remember its outer variables and can access them. All functions in javascript are closures.

Function bundled with its lexical environment is known as a closure. Whenever function is returned, even if it is vanished in execution context but it still remembers the reference it was pointing to. Its not just that function alone it returns but the entire closure.

```
function outer(b) {
    function inner() {
        console.log(a, b);
    }
    let a = 10;
    return inner
}
var close = outer("Hello");
close();
```

'inner' function forms a closure with its outer environment and 'b' is also a part of 'outer' environment of 'inner' function. So it forms close with 'b' also.

→ Closures helps in data hiding and encapsulation.

$$\begin{tikzcd}[column sep=small] & \text{Types of Programming Language.} \arrow[dl] \arrow[dr] & \\ \text{Imperative} & & \text{Declarative} \end{tikzcd}$$

→ **Imperative :—**

Imperative lang. are those languages in which we manually mention every step algorithm step of the algorithm what to do, how to do, where to do.

→ **Declarative :—**

In declarative, we just tell what we need and every thing is sorted by the language.

* **Iterators :—**

In javascript an Iterator is an object which defines a sequence and potentially a return value upon its termination.

Specifically, an iterator is any object which implements the iterator protocal by having a next() method that returns an object with two properties.

* **Generators :—**

Generator functions allows us to define an iterative algorithm by writing a single ~~algorithm~~ function whose execution is not continous.

Generator functions are written using function* syntax.

Page No.

Date / /

When ~~we~~ called, generator functions do not initially execute their code. Instead they return a special type of iterator, called a Generator.

When a value is consumed by calling the generator's next method the Generator function executes until it encounter the yield keyword.

→ The execution of the generator function doesn't start at the time of calling the function.

→ The moment we got yield, the execution of the generator function stop there. and whatever we was yielding, it returns ~~th~~ till that value.

→ 'yield' is similar to return but not a return