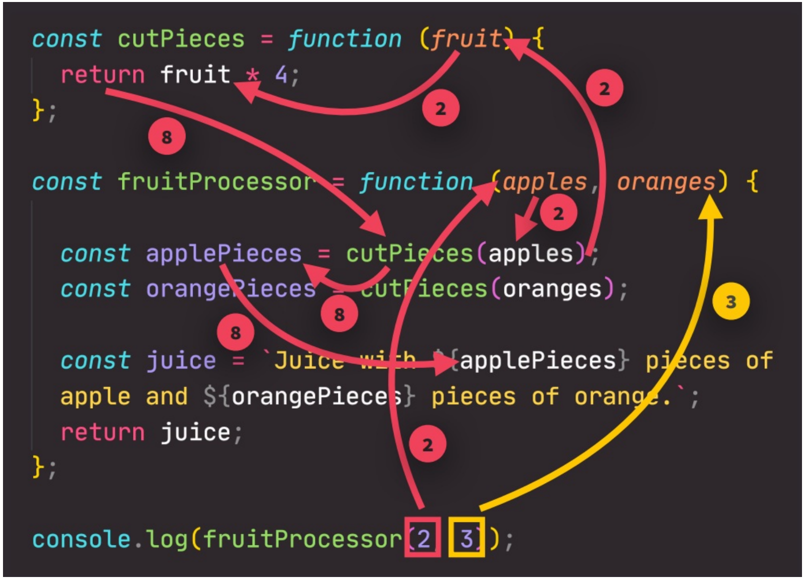


```
const cutPieces = function (fruit) {
  return fruit * 4;
};

const fruitProcessor = function (apples, oranges) {

  const applePieces = cutPieces(apples);
  const orangePieces = cutPieces(oranges);

  const juice = `Juice with ${applePieces} pieces of
  apple and ${orangePieces} pieces of orange.`;
  return juice;
};

console.log(fruitProcessor(2, 3));
```

* **Functions** : (Abstraction ⟶ hiding Implementation details)

What ?

how to use ?

how they work ↗ code ✓ ↘ internally (JS) v.imp ✗

→ Solves the problem of DRY, provides reusability

→ Can take Inputs in form arguments

→ Can give ouputs / result using return keyword

→ return shutdowns the function, any after return will not be executed, you can send any values with return.

9:15pm - 9:30pm
BREAK

→ we used many functions without knowing them,

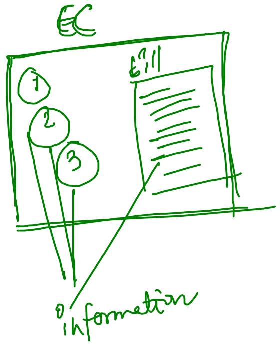
```
const a = Number("123")        fromCharCode( )
const s = String(123)          CharCodeAt( )
const b = parseInt(5/10)       console.log( )
```

✯ Execution Context :

→ It is like a box, where a piece of JS code is executed

It contains all info for the code to be executed

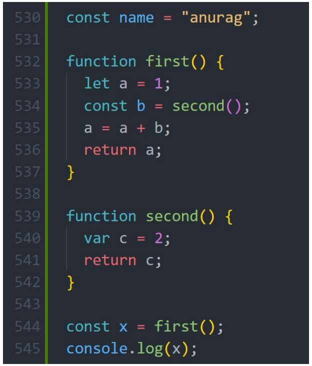
1. shoes
2. earphones
3. iphone



1. variables
   - → let, const, var
   - → function
   - → parameters / arguments
2. ~~Scope Chain~~

→ only one g EC is created per program

→ EC is created for every function



```
const name = "anurag";

function first() {
  let a = 1;
  const b = second();
  a = a + b;
  return a;
}

function second() {
  var c = 2;
  return c;
}

const x = first();
console.log(x);
```

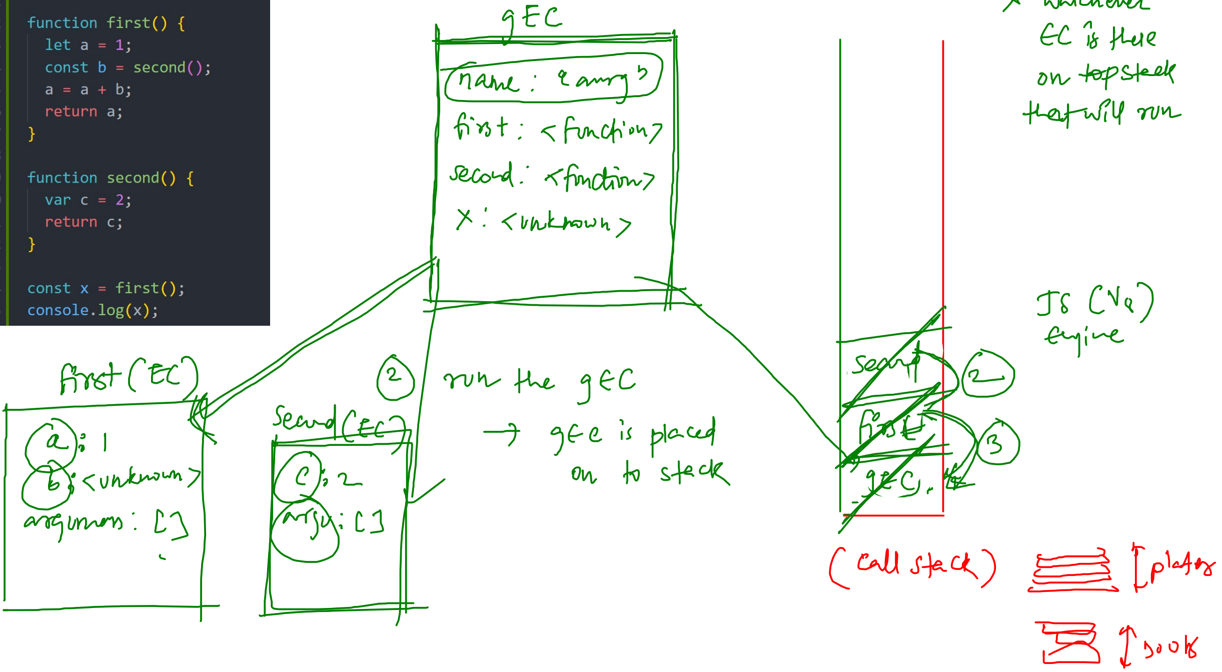
① A global Execution Context is created

② run the gEC

→ gEC is placed on to stack

★ whichever EC is there on ~~top~~ stack that will run

JS (V8) engine



Global scope

```
const a = 5;
let fname = "arjun";
let year = 2023;
```

→ outside of a function or block

→ They are accessible anywhere in the code.

function scope

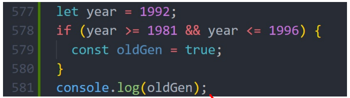
```
function calcAge(year){
    year = year + 1;
    const age = 2023
                - year;
    return age;
}
const year = 2000;
console.log(calcAge(year));
console.log(year);
console.log(age);
```

reference Error

→ variable are accesible inside the function only.

→ also called as local scope.

block scope



```
let year = 1992;
if (year >= 1981 && year <= 1996) {
  const oldGen = true;
}
console.log(oldGen);
```

reference Error

→ variables are accessible only inside block (if, else, else if, for, while, ~~do while~~)

→ this applies only to let and const variables (Twist)

→ var is only function scope ~~and~~ not block scoped due to hoisting.