★ Scope :

global scope

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

function test() {
  console.log(a, fname, year);
}

console.log(a, fname, year);
```

→ declared outside of a function or block

→ They are accessible everywhere in code.

function scope

```
function calcAge(birthYear) {
  const currYear = 2023;
  const age = currYear - birthYear;
  return age;
}

console.log(calcAge(2001));
console.log(currYear, age);
```

Reference error : not defined

→ variables are accessible only inside the function.

→ also called as local scope.

block scope

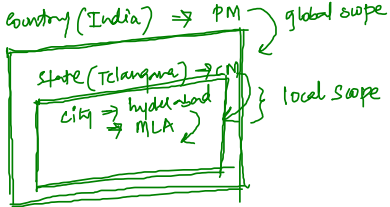
```
const birthYear = 1991;
if (1981 <= birthYear && birthYear <= 1996) {
  const oldGen = true;
}
console.log(oldGen);
```

Reference error : not defined

→ variables are accessible only inside if/else/else if/for/while.

→ functions are kind of block scoped.

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




```
// function scope
function calcAge(birthYear) {
  const currYear = 2023;
  var age = currYear - birthYear;
  return age;
}

console.log(calcAge(2001));
console.log(age); // as expected, reference error

// block scope
const birthYear = 1991;
if (1981 <= birthYear && birthYear <= 1996) {
  const oldGen = true;
  var century = 20;
}
console.log(century); // var type variables are not block scoped
console.log(oldGen); // reference error
```

* Hoisting :

→ It makes variables and functions accessible / usable in the code before they are actually declared. "variables / functions are lifted to the top of their scope".

↓ behind the scenes

Before execution, code is scanned for variable / function declarations and it decides whether to lift or not.

functions → Yes

var declarations → Yes, undefined

let, const declarations → NO, ← Temporal Dead Zone ? (zone where the variable is not accessible)

↓

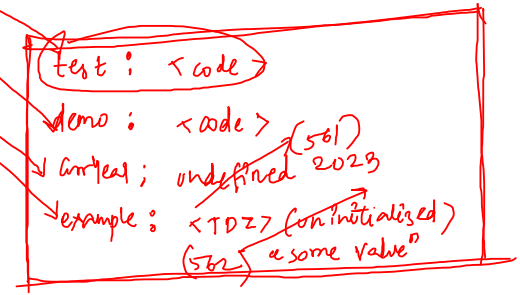
but technically kind of yes (uninitialized)

```
test();
demo();
console.log(currYear); // undefined
console.log(example); // udefined

function test() {
  console.log("Hi, How are you ?");
}

function demo() {
  console.log("demo function");
}

var currYear = 2023;
let example = "some value";
```

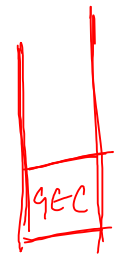
★ It is all because of EC,

Hi how are you
demo function
undefined

① GEC



② put it to stack



★ These things lead to hoisting

test demo

Eg ①:

```
563  y = 3;
564  console.log(y);   → 3
565  var y = 100;
566  console.log(y);   → 100
```

* If you change
  var y = 100;
  to let y = 100;
  (reference error)
  (let is not hoisted)

① GEC creation (declarations)

y : ~~undefined~~ 3 (563)
100 (565)

② running GEC,

(563) y = 3
(564) 3
(565) y = 100
(566) 100

var is hoisted

If let y = 100; (565)

① y: <TDZ>

② (563) y = 3
cannot access y before initialisation.

let is not hoisted
const (they will be hoisted but TDZ)

9:30 - 9:45 pm
BREAK

```
568  function first() {
569      console.log("first");
570      second();
571  }
572
573  function second() {
574      console.log("first");
575  }
576
577  first();
```

op: first
second

①

first:
second:

first EC

second EC

first
GEC

Eg 2 :

```
563  function example() {
564    console.log(a);
565  }
566  console.log(a);
567  var a = 1;
568  example();
```

① GEC

example: <code>

a: ~~undefined~~ → 1

② (566) undefined

(567) a = 1

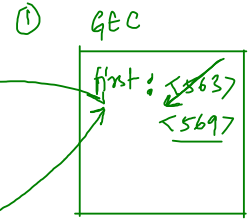
(568) 1

↘ (564)

* a has a global hence it is accessible inside example.

Eg ③ :

```
563  function first() {
564    console.log(1);
565  }
566
567  first();
568
569  function first() {
570    console.log(2);
571  }
572
573  first();
```



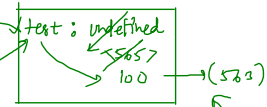
② (567) 2

(513) 2

* If you same declarations they will be overridden for the lattest (top → bottom) (latest)

Q4 :

```
563  var test = 100;
564  console.log(test);
565  function test() {
566    console.log("Inside function test");
567  }
568  console.log(test);
```



① test — undefined — during creation (563)

② test — { code } — during creation (565)

③ test — 100 — during run/execution (563)

② Execution/run
(563) test = 100
(564) 100
(568) 100

④ (565) is a declaration which already handled in creation phase hence do not touch it.

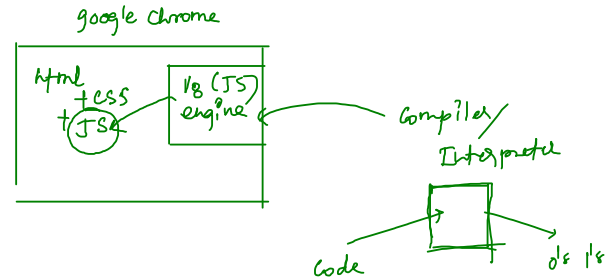
★ process.stdout.write (horizontal printing) :

→ It doesn't work inside your browser.

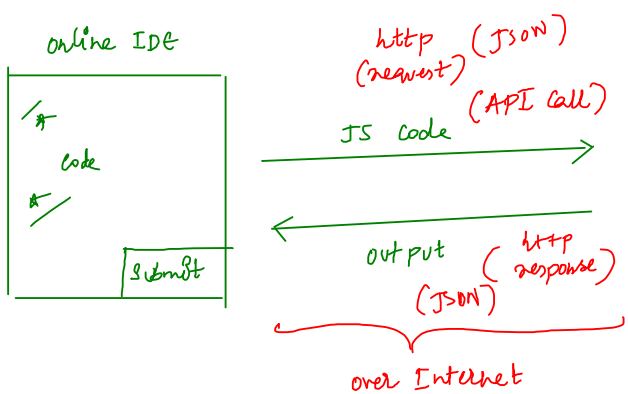
C/C++ / Java / Python / JS

mingw	JDK	Python	Web browser

① Install compiler

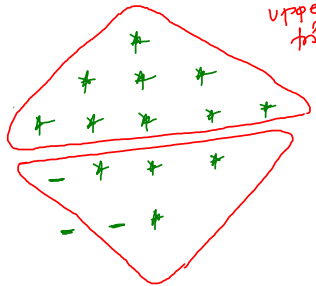


→ It works with node.js environment,
process.stdout.write is not a JS function,
it is provided by node.JS.

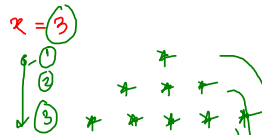

Online IDE
code
submit
http (JSON)
(request)
(API call)
JS code
output
(http response)
(JSON)
over Internet
Backend server (some other computer over internet)
The code sent from IDE will run here and op will be sent back.
V8 Engine
JS code is run using node.JS on the server
how will you get v8 engine on some random backed machine
(Node.JS)
(runtime Environment for JS)
JSON - JavaScript object notation
later modules

★ Diamond pattern :

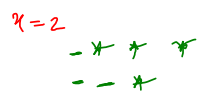
Eg: n = 5

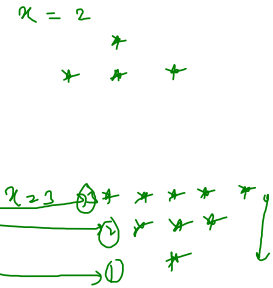


upper triangle



lower triangle





n = 5 → upperTriangle(?) ③ ⇒ PI(5/2)+1

n = 5 → lowerTriangle(?) ② ⇒ PI(5/2)

★ solving upper triangle,

x = 4

rows = x

cols = spaces + stars

① ⇒ spa = 4-1 = 3

② ⇒ spa = 4-2 = 2

① ⇒ star = 2(1)-1 = 1

② ⇒ star = 2(2)-1 = 3

③ ⇒ star = 2(3)-1 = 5

	(x-row) Spaces	(2*row-1) Stars
row = ① ⇒	3	1
② ⇒	2	3
③ ⇒	1	5
④ ⇒	0	7

* lower Triangle,

x = 2

```
- * * *
- - *
```

x = 3

```
- * * * * *
- - * * *
- - - *
```

x = 4

```
- * * * * * * (1)
- - * * * * * (2)
- - - * * * (3)
- - - - * (4)
```

cols = spaces + stars

spaces = row

row - x
= 4

	spaces	stars = (2 * (x - row) + 1)
(1) ⇒	1	7
(2) ⇒	2	5
(3) ⇒	3	3
(4) ⇒	4	1