**### steps involved**

1. **tokeninzing/lexing** - breaks expression into tokens.
2. **parsing** – creates AST abstract syntax tree - it does hoisting, find syntax errors and some early errors.
3. **executing** - executes the AST - converting AST into machine level instructions
4. machine understand these instructions, schedule them on CPU as per the scheduling strategy like round robin.

code -> js engine tokenize -> js engine parsing -> js engine understanding the AST and converting it into byte code -> instructions goes to the CPU

memory allocation - it is done by the underlying processor

**\*\*JS engine converts the code into machine language that can be processed by the CPU\*\***

**### targets**

-- a value is assigned to a target

```js

var name = "yati"

// var name is declaration handled at compiled time only

// name="yati" is assignment, done at execution time

```

```js

function getStudentName(studentID) {} // function declaration doen at compile time

```

```js

var getStudentName = function(studentID) {} /\* function is a special case, assignment is

also done at compile time and it is called function hoisting

\*/

```

**\*\*scopes are identified during compilation and are created during runtime\*\***

Compiler looks for a variable into its scope if not available then to it’s parent scope and then it’s parent. This way it keeps going till the window scope(the parent of all)

**### compile time –**

->scopes are determined/identified

-> variables are labled with their scopes.

**\*this labeling of variables is used at the runtime to create their scopes\***

**Scope Manager**collects and maintain a look up list of all variables to maintain their scope

**Compiler**  
parsing and code generation

JS Engine

**\*\*scope may also change at runtime by using eval and with. eval creates scope at runtime by executing the javascript string\*\***

**### At the beginning of a scope**

* identifier of a function declaration gets assigned to the associated function

function getName(){}

\*getName will be assigned to it’s function refernce at the beginning of the scope\*

* variables declared with **var** get assigned to **undefined**. This assignement brings them on to the stack for use.

// at the beginging of the scope js engine asks the scope manager for variable **fname**  and assigns it to **undefined**

console.log(fname); // undefined

var fname = 'yati';

* variables declared with **let/const** remains in **temporal dead zone** until its full declaration-and-initialization are executed.

**\*\*\*There are separate instances of *scope manager* for every scope.\*\*\***