Components of Execution Context

There are two important components of an execution context: the Variable Environment and the Thread of Execution.

Variable Environment

1. The Variable Environment is a fundamental component that organizes and holds all the variables, functions, and parameters accessible within a given scope.

2. It is created when a new function is executed and contains information about all the variables that are declared within that function and the values assigned to them.

3. The variable environment also includes a reference to the outer environment, which is the variable environment of the parent scope.

Thread of Execution

1. The Thread of Execution is the sequence of code execution that is currently being executed.

2. It is responsible for running the code one line at a time and keeping track of where the execution is at any given moment.

3. When a new function is called, a new thread of execution is created, and the execution continues within that thread until the function returns.

Global vs Local Execution Context

Global Execution Context

1. The global execution context is the default environment in which JavaScript code is executed.

2. The global execution context is created when the JavaScript program starts running and stays in memory until the program ends.

3. During the creation phase, the JavaScript engine performs several steps to set up the environment for executing the code. These steps are described below:

    1. Defining Window Object
       The engine defines the global window object, which serves as the outermost object in the environment.

    2. Creating this Variable
       The this variable is created and assigned to the window object. It represents the context in which the current code is executing.

    3. Hoisting
       Hoisting takes place, where variable and function declarations are moved to the top of their respective scopes. This allows you to use variables and functions before they are formally declared in the code.

    4. Memory Allocation
       After hoisting, the engine allocates memory for variables and functions, preparing them for later use during the execution phase. It's important to note that the way variables are allocated and their initial values can vary depending on the type of declaration.

        a) var Variables
           When a variable declared with var is encountered during memory allocation, it is assigned the default value of undefined. This means that although the variable exists in memory, it holds the value undefined until a value is explicitly assigned to it.

        b) let and const Variables
           Variables declared with let and const also go through memory allocation. It's during this phase that let and const variables are assigned the initial value of undefined within the TDZ.

Local Execution Context

1. A local execution context is created each time a function is called. It serves as a separate environment for the function's execution, encompassing several important aspects.

2. Firstly, the local execution context defines the this variable. The value of this is determined based on how the function is invoked. If the function is called in the global scope or without any specific context, this will be assigned the global `window` object. However, in strict mode, this will be undefined in such cases.

3. Additionally, the local execution context includes the creation of the `arguments` object. This object is available within the function and contains a list of all the arguments passed to it.

4. Furthermore, during the creation phase of the local execution context, memory allocation takes place similar to the one we discussed above.

5. When a variable is referenced in a function, JavaScript first looks for it in the local execution context's variable environment. If the variable is not found there, it looks in the parent execution context (if any), and so on, until it reaches the global execution context.

6. When a function completes its execution, its local execution context is removed from memory.

```
var userName='Tom';
var userAge=10;
console.log(`username: ${userName}`);
console.log(`userage: ${userAge}`);

function greetUser(name){
  var greet='I hope you are doing fine.';
  console.log(`hello, ${name}, ${greet}`);
  var currentYear = 2030;
  const year = currentYear - userAge;
  return `Your birthyear is ${year}`;
}
const birthYear = greetUser(userName);
console.log(birthYear);
```

Global Execution

```
1  var userName='Tom';
2  var userAge=10;
3
4  console.log(`username: ${userName}`);
5  console.log(`userage: ${userAge}`);
6
7  function greetUser(name){          Local Execution
8    var greet='I hope you are doing fine.';
9    console.log(`hello, ${name}, ${greet}`);
10   var currentYear = 2030;
11   const year = currentYear - userAge;
12   return `Your birthyear is ${year}`;
13 }
14 const birthYear = greetUser(userName);
15 console.log(birthYear);
```

Creation Phase

During the creation of the global execution context, the JavaScript engine declares three variables (userName, userAge, and birthYear) and initializes them to undefined. It also declares a function named greetUser, which is stored in memory but not executed yet.

Global Execution Context

Variable Environment	Thread of Execution
userName : undefined	
userAge : undefined	
greetUser : f greetUser (name)	
birthYear : undefined	

Execution Phase:

1. The JavaScript engine assigns 'Tom' to the userName variable and 10 to the userAge variable. It then executes console.log() twice, outputting the values of userName and userAge to the console as "username: Tom" and "userAge: 10" as shown below:

Global Execution Context

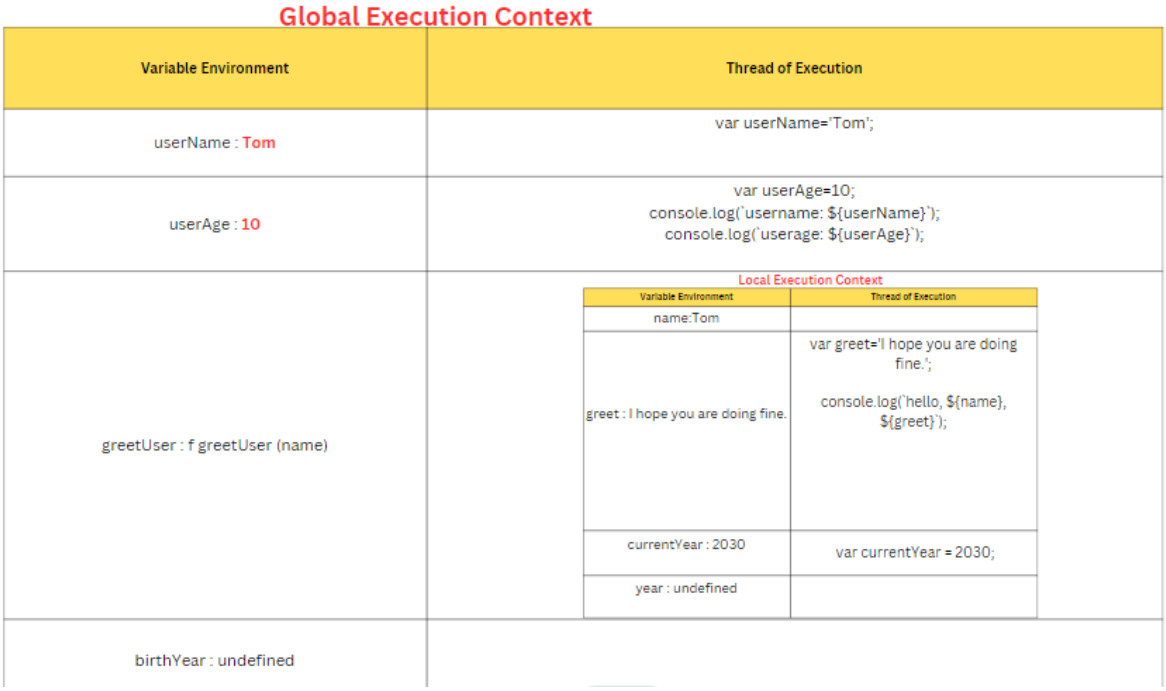
Variable Environment	Thread of Execution
userName : Tom	var userName='Tom';
userAge : 10	var userAge=10;
greetUser : f greetUser (name)	
birthYear : undefined	

2. When a function is invoked, a new Execution Context is built all together to carry out the same procedures for that function call/invoke. The JavaScript engine creates a new execution context (as shown below) for the greetUser function following the same procedure discussed above.

Global Execution Context

<table>
<tr><th>Variable Environment</th><th>Thread of Execution</th></tr>
<tr><td>userName : Tom</td><td>var userName='Tom';</td></tr>
<tr><td>userAge : 10</td><td>var userAge=10;
console.log(`username: ${userName}`);
console.log(`userage: ${userAge}`);</td></tr>
<tr><td>greetUser : f greetUser (name)</td><td>Local Execution Context
<table>
<tr><th>Variable Environment</th><th>Thread of Execution</th></tr>
<tr><td>name:Tom</td><td></td></tr>
<tr><td>greet : undefined</td><td></td></tr>
<tr><td>currentYear : undefined</td><td></td></tr>
<tr><td>year : undefined</td><td></td></tr>
</table></td></tr>
<tr><td>birthYear : undefined</td><td></td></tr>
</table>

A) The further workflow of the Local Execution Context is described in the following figure, where the variables within the greetUser function are assigned values.



B) Finally, The greetUser function returns a string containing the calculated birth year and the local execution context for the greetUser function is removed, and control is returned to the global execution context.