**SCOPE**

* Scope là phạm vi truy cập của biến, hàm,…
* Có 3 loại phạm vi
  + Global: phạm vi toàn cầu
  + Code block: phạm vi khối code
  + Local scope: phạm vi phần than của code
* Global: khi khai biến, hàm,… ở đâu thì bất kì chỗ nào thì các biến, hàm,… đều có thể truy cập được ở mọi nơi.
* Code block: khi khai báo biến let, const thì những biến này truy cập được ở trong khối code đó ngoài khối không truy cập được
* Local scope: khi khai báo biến, hàm,… nằm trong một hàm (hàm con) mà hàm đó lại nằm trong một hàm bên ngoài (hàm cha) nên những biến, hàm đó chỉ được sử dụng trong hàm con còn bên ngoài hàm cha không được sử dụng.
* VD (global):

|  |
| --- |
| var message = “Hoc ve scope” //biến global  console.log(message)  function logger(){  console.log(message)  }  logger() |

|  |
| --- |
| function log() { //Hàm global  console.log(“In ra log”)  }  function logger() {  log()  }  logger() |

* VD (code block), thông thường là cặp { }:

|  |
| --- |
| {  const age = 18 //Biến chỉ được sử dụng trong code block  console.log(age) //Thực hiện được  }  console.log(age) //Báo lỗi, không thực hiện được  {  let name = ‘Nguyen Van A’ //Biến chỉ được sử dụng trong code block  console.log(name) //Thực hiện được  }  Console.log(name) //Báo lỗi, không thực hiện được |

* Trường hợp đặc biệt: if-else hay for mọi thứ nằm bên trong nếu thoả mãn gì đó mà chỉ có duy nhất 1 câu lệnh thì không cần phải { } nhưng nếu bên trong đó là khai báo biến bằng let hay var thì bên ngoài cũng không thể nào truy cập được.
* VD: (local scope)

|  |
| --- |
| function logger() {  var fullName = ‘Nguyen Van A’  console.log(fullName) //Thực hiện được  }  logger()  console.log(fullName) //Báo lỗi |

* Khi gọi mỗi hàm luôn có 1 phạm vi mới được tạo ra. Thậm chí nếu có 1 hàm mà gọi đến nhiều lần thì hàm đó sẽ có nhiều phạm vi truy cập. Tính từ thời điểm bắt đầu gọi hàm đầu tiên thì hàm sẽ có 2 phạm vi truy cập lần đầu là lúc tạo hàm và lần 2 sẽ là lúc gọi hàm lần đầu, nếu có gọi nhiều lần sẽ có nhiều hơn 2 phạm vi truy cập.
* VD:

|  |
| --- |
| function logger(first, last) {  console.log(first, last)  }  logger(‘Nguyen’, ‘A’)  logger(‘Tran’, ‘B’)  logger(‘Ha’, ‘C’) |

* Giải thích: Ở mỗi lần gọi hàm đều truyền 2 tham số và 2 tham số ở mỗi lần gọi là khác nhau do có thêm một phạm vi truy cập mới nên gần như 2 tham số first, last đều có thể nhận bất kì giá trị nào (chuỗi, số, object,…) ở mỗi lần gọi.
* Các hàm có thể truy cập các biện được khai báo trong phạm vi của nó và bên ngoài nó.
* VD:

|  |
| --- |
| function logger(first, last) {  console.log(first, last)  const age = 20  console.log(age)  }  logger(‘Nguyen’, ‘A’)  logger(‘Tran’, ‘B’)  logger(‘Ha’, ‘C’) |

* Giải thích: biến nằm trong một hàm thì hàm có quyền truy cập biến đó.
* VD:

|  |
| --- |
| const age = 16  function logger() {  function logger2() {  console.log(age)  }  logger2()  }  logger() |

* Giải thích: biến nằm bên ngoài nên mọi hàm có thể truy cập được cho dù nó ở bên trong thì vẫn có thể truy cập biến global. (tương tự như hàm)
* Mở rộng: thông thường mỗi lần gọi hàm thì hàm sẽ tạo một phạm vi truy cập mới, tương tự với biến nếu như biến khai báo ở nhiều vị trí khác nhau trong code block và trùng tên thì nó cũng tạo ra một phạm vi truy cập mới cho biến đó và nếu lấy giá trị của biến đó, nó sẽ lấy giá trị thay đổi gần nhất.
* VD:

|  |
| --- |
| const age = 18  {  const age = 16  {  {  const age = 14  {  const age = 12  }  console.log(age) //Output là 14  }  console.log(age) //Output là 16  }  } |

* Lưu ý: nếu cùng tên và cùng phân cấp thì nó sẽ báo lỗi
* Một số trường hợp nếu biến đó khai báo sau khi sử dụng thì nó sẽ báo lỗi.
* VD:

|  |
| --- |
| const age = 14  {  {  {  console.log(age) //Báo lỗi vì không thể sử dụng biến trước khi khai báo  const age = 12  }  }  } |

* Giải thích: khi in biến age nó sẽ quét tìm age bên trong code block trước nếu không có nó sẽ tìm dần ra ngoài cho đến khi tìm được biến age. Trong trường hợp này biến age khai báo sau khi sử dụng nên dẫn đến lỗi.
* Lỗi này cũng tương tự với hàm thay vì chỉ có biến.
* Biến bị xoá khỏi bộ nhớ:
  + Biến toàn cầu: khi khai báo biến, giá trị của nó sẽ lưu trữ trong bộ nhớ và nó sẽ xoá khi tắt trang website hoặc reload trang website.
  + Biến trong code block và trong hàm (let hoặc const): theo cơ chế biến đó khai báo trong code block bằng let hoặc const thì chỉ sử dụng trong code block đó và bên ngoài không truy cập được khi ra khỏi code block nó sẽ giải phóng biến khỏi bộ nhớ (phương pháp này để tối ưu)