**Bài 01: ECMAScript 2015 - ES6 là gì?**

Hôm nay chúng ta bắt đầu tìm hiểu một thứ mới hơn về Javascript vừa được ra mắt vào năm 2015 đó là **ECMAScript**, đây có thể coi là được một phiên bản nâng cấp mới nhất dành cho Javascript nên phải học nó để khỏi tụt hậu phải không các bạn :)

Trước tiên chúng ta sẽ tìm hiểu định nghĩa của ES6 trước.

**1. ECMAScript ES6 là gì?**

ES6 là chữ viết tắt của **ECMAScript 6,** đây được coi là một tập hợp các kỹ thuật nâng cao của Javascript và là phiên bản mới nhất của chuẩn **ECMAScript**. ECMAScript do hiệp hội các nhà sản xuất máy tính Châu Âu đề xuất làm tiêu chuẩn của ngôn ngữ Javascript. Bạn cứ nghĩ xem hiện nay có khá nhiều trình duyệt Browser ra đời và nếu mỗi Browser lại có cách chạy Javascript khác nhau thì các trang web không thể hoạt động trên tất cả các trình duyệt đó được, vì vậy cần có một chuẩn chung để bắt buộc các browser phải phát triển dựa theo chuẩn đó.

ES6 ra đời vào năm 2015 nên cái tên **ES2015** được lấy làm tên chính thức với nhiều tính năng mới, học hỏi các ngôn ngữ cấp cao khác. Hy vọng dần theo thời gian Javascript trở thành một ngôn ngữ lập trình hướng đối tượng.

Phiên bản sắp ra trong năm 2017 đó là phiên bản ES7 cũng đang được nghiên cứu và phát triển, họ cũng nhắm đến các kiến thức mới lạ như *async function, observer*, .. Hy vọng sẽ có nhiều biến động mới.



**2. Các chức năng mới của ES6**

Bài này mình chỉ giới thiệu các chức năng mới mà ES6 đã tích hợp vào ngôn ngữ Javascript, vì vậy sẽ không có các ví dụ kèm theo mà thay vào đó sẽ được viết ở một bài cụ thể gần nhất.

Sau đây là một số chức năng mới thêm vào trong ES6.

* **Block Scoped**: Định nghĩa biến với từ khóa let, cách định nghĩa này thì biến chỉ tồn tại trong phạm vi khối của nó (Block Scope)
* **Destructuring Assignments**: Bạn có thể khởi tạo các biến từ một mảng  bằng một dòng code đơn giản.
* **Default Parameters**: Bạn có thể gán giá trị mặc định cho các tham số.
* **Rest Parameter**: Tham số không giới hạn
* **Arrow function**: Bạn có thể tạo hàm bằng cách sử dụng dấu mũi tên =>.
* **Template String:**Tạo templaet HTML cực kì đơn giản
* **Weak, Set:**các kiểu dữ liệu phức tạp mới
* **Iterables và iterators**
* **Class, import**
* ...

Còn khá nhiều tính năng nhưng mình không liệt kê thêm, bạn có thể theo dõi toàn bộ series này để hiểu rõ hơn. Các tính năng mới của ES6 được đánh giá tương đối khó học, vì vậy nếu bạn không vững các kiến thức [Javascript căn bản](http://freetuts.net/hoc-javascript/javascript-can-ban) thì bạn không thể học ES6 được.

Hiện nay các JS Framework như NodeJS, Angular2, React Native ... đều sử dụng ES6 nên để học được các framework đó thì ban phải thành thạo Javascript nói chung và ES6 nói riêng.

**3. Một số phiên bản khác của ES6**

ES6 là phiên bản mới nhất tính đến thời điểm hiện tại là tháng 1 năm 2017, các phiên bản khác của ES6 bao gồm:

* **ECMAScript 5 (December 2009)**: Phiên bản này hầu hết các trình duyệt đều hỗ trợ, nó cải tiến và bổ sung thêm một số thư viện chuẩn, sử dụng chế độ [strict mode](http://freetuts.net/use-strict-la-gi-strict-mode-trong-javascript-407.html) nghiêm ngặt.
* **ECMAScript 5.1 (June 2011)**: ES5 đã được xem như là một chuẩn ISO, phiên bản này đã sửa một số lỗi nhỏ.
* **ECMAScript 6 (June 2015)**: Là phiên bản hiện tại, bổ sung nhiều cú pháp và thư viện.
* **ECMAScript 2016**: Hay còn gọi là ES7, đây là phiên bản tương lai chưa được public, được hứa hẹn có nhiều bổ sung giúp Javascript trở thành ngôn ngữ sử dụng được hầu hết ở mọi môi trường.

**4. Lời kết**

Nói về lý thuyết thì dài dòng và khó hiểu nên mình không trình bày nhiều về lý thuyết nữa để tránh sử dụng những từ không đúng, hy vọng qua series này sẽ giúp ích các bạn trong việc nghiên cứu ES6 để làm tiền đề học ES7 sắp ra đời.

**Bài 02: Điều kiện cần để tự học ES6**

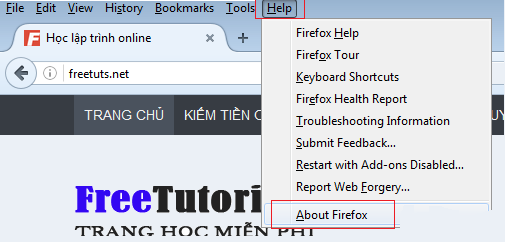
ES6 được ra đời vào năm 2015 nên có thể coi nó là một phiên bản mới nhất của Javascript, vì vậy có thể các trình duyệt ra đời trước năm 2015 sẽ không hoạt động được. Nhưng bạn cũng đừng lo lắng quá bởi vì các Version mới nhất của Chrome và Firefox hiện nay hầu hết đã được tích hợp ES6 nên bạn hoàn toàn yên tâm về việc cài đặt môi trường ES6.

Nếu bạn chưa cài đặt browser nào thì hãy lên trang chủ của nó, download version mới nhất về và cài đặt là xong. Còn bạn đang sử dụng browser quá lỗi thời thì hãy thực hiện nâng cấp nó theo các hướng dẫn dưới đây.

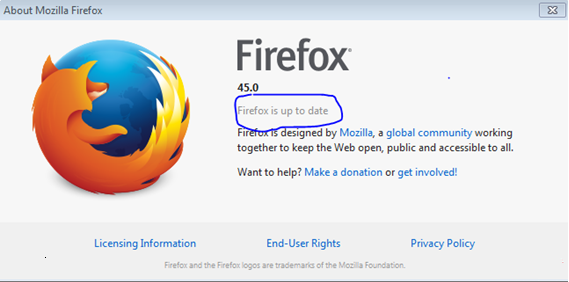
**1. Update Firefox để học ES6**

Vậy công việc bây giờ của bạn là hãy cập nhật phiên bản mới nhất của Firefox để học bài nhé.

Trên thanh công cụ bạn chọn **Help**-> **About Firefox**. Nếu không thấy thanh công cụ thì bạn nhấn phím alt nhé.



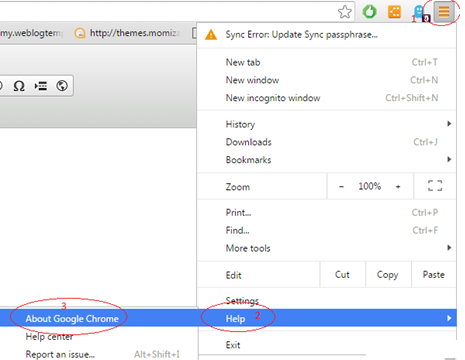
Một ô cửa sổ xuất hiện và bạn vui lòng chờ nó tự động cập nhật, sau khi cập nhật xong nó sẽ bắt bạn reload lại là coi như đã thành công.



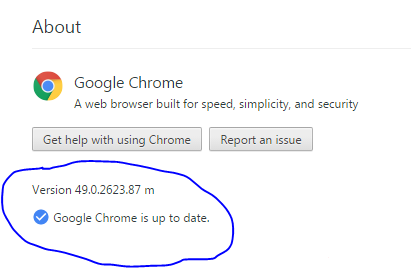
**2. Update Chrome để học ES6**

Nếu bạn sử dụng Chrome thì hãy nâng cấp version mới nhất bằng cách thực hiện các bước sau.

Tại menu công cụ bạn chọn **Help** -> **About Google Chrome**.



Một tab mới mở ra và bạn cũng chờ nó tự động cập nhật nhé.



Ok vậy là bạn đã có thể bắt đầu học ES6 được rồi đấy :)

**3. Lời kết**

Bài này khá quan trọng đấy, vì nếu bạn không cập nhật trình duyệt thì bạn sẽ không thể chạy code của ES6 được. Tuy vẫn có cách sử dụng thư viện compier nhưng mình thấy đó là điều không cần thiết. Và các bạn thấy rõ ràng các trình duyệt version mới nhất đã tích hợp ES6 vào bộ thư viện của họ, vì vậy bạn hoàn toàn yên tâm mà học nhé, không sợ học rồi không biết ứng dụng vào đâu.

**Bài 03: ES6 Block Scoped - Khởi tạo biến với từ khóa let**

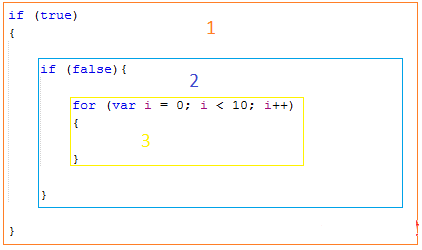
Trong ES6 có cung cấp một từ khóa nữa dùng để khởi  tạo biến đó là từ khóa let, từ khóa này khác với từ khóa var ở chỗ phạm vi hoạt động. Với từ khóa var nếu ban khai báo biến bên trong hàm thì đó là biến cục bộ, còn nếu bạn khai báo bên ngoài hàm thì nó sẽ là một biến toàn cục. Còn với từ khóa let thì phạm vi hoạt động của nó nhỏ hơn, nó chỉ tồn tại bên trong khối đang khai báo và ta gọi đây là phạm vi **block scoped**.

Trước tiên mình sẽ giới thiệu từ khóa **Block Scoped** đã nhé.

**1. Block Scoped là gì?**

Block Scoped là phạm vi trong một khối, nghĩa là chỉ hoạt động trong phạm vi được khai báo bời cặp {}.

Xem hình dưới đây để hiểu rõ hơn.



Như trong hình thì vị trí 1 là một block, vị trí 2 là một block và vị trí 3 cũng là một block. Nếu để ý kỹ hơn nữa thì bạn thấy vị trí 1 sẽ bao trùm cả vị trí 2 + 3 và vị trí 2 bao trùm cả vị trí 3. Vậy Block Scoped là phạm vi chứa tất cả những đoạn code nằm bên trong cặp thẻ {}.

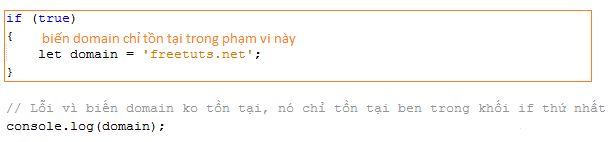
**2. Dùng từ khóa let để khai báo biến trong ES6**

**Lưu ý**: Khi sử dụng ES6 thì bạn nên bật chế độ [strict mode](http://freetuts.net/use-strict-la-gi-strict-mode-trong-javascript-407.html), chế độ này tạm gọi là chế độ nghiêm ngặt trong việc sử dụng cú pháp của Javascript.

Như ở phần giới thiệu, từ khóa let dùng để khởi tạo một biến nhưng biến đó chỉ có tác dụng bên trong khối đang khai báo (*block-scoped*).

**Cú pháp**: let var\_name = var\_value;

Ở hình dưới đây mình khai báo biến domain và biến này chỉ hoạt động trong phạm vi của lệnh if.



**3. Khi nào nên sử dụng let để khai báo biến**

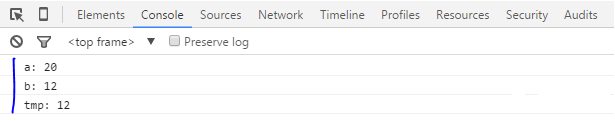
Với phạm vi hoạt động hẹp như vậy thì let thường dùng để khai báo các biến mang tính chất tạm thời, nghĩa là nó chỉ sống trong một phạm vi hoạt động của khối đó thôi, không sử dụng qua vị trí khác.

**Ví dụ**: Viết chương trình hoán đổi giá trị của hai biến a và b nếu giá trị của a nhỏ hơn giá trị của b.

Với bài toán này thì thông thường ta phải khai báo thêm một biến tạm như sau:

|  |
| --- |
| var a = 12;  var b = 20;  if (a < b)  {      var tmp = a;      a = b;      b = tmp;  }  console.log("a: " + a);  console.log("b: " + b);  console.log("tmp: " + tmp); |

Sử dụng [firebug](http://freetuts.net/su-dung-firebug-de-kiem-tra-loi-javascript-380.html) bạn sẽ thấy kết quả như hình sau:

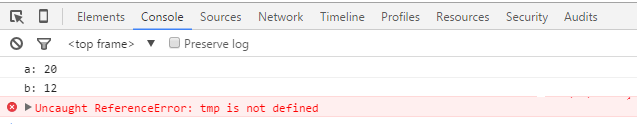


Như vậy biến tmp sau khi kết thúc lệnh if nó vẫn tồn tại => dư thừa không cần thiết.

Quay lại bài toán ta sử dụng từ khóa let như sau:

|  |
| --- |
| var a = 12;  var b = 20;  if (a < b)  {      let tmp = a;      a = b;      b = tmp;  }  console.log("a: " + a);  console.log("b: " + b);  console.log("tmp: " + tmp); // đoạn này lỗi |

Đoạn code này chạy bình thường, tuy nhiến nếu bạn cố ý console.log(tmp) như ví dụ trên thì sẽ xuất hiện lỗi biến tmp không tồn tại vì biến tmp chỉ tồn tại bên trong lệnh if (*xem hình*).



**4. Lời kết**

Rõ ràng việc sử dụng từ khóa let có những ưu điểm và nhược điểm khác nhau, vì vậy không phải thấy nó mới được bổ sung vào là sử dụng đâu nhé :). Ví dụ nếu bạn sử dụng một biến cho toàn ứng dụng và biến đó thay đổi liên tục thì bắt buộc phải sử dụng var để khai báo.

**Bài 04: Arrow Function trong ES6**

Trong bài này chúng ta tìm hiểu một cách định nghĩa function mới bằng cách sử dụng dấu mui tên =>, cách này được thêm vào bộ ES6 với cú pháp hoàn toàn mới lạ.

Trong Javascript để tạo một function thì thông thường chúng ta sử dụng hai cách sau:

|  |
| --- |
| // Cách 1  function Name(var1, var2){  }  // Cách 2  var Name = function(var1, var2,){  } |

Với ES6 thì bạn có thêm một cách đó là sử dụng dấu mũi tên => rất phức tạp, chi tiết thế nào thì chúng ta cùng tìm hiểu nhé.

**1. Sử dụng Arrow function trong ES6**

Nói là ES6 nhưng thực chất hầu hết các trình duyệt hiện nay đều đã hỗ trợ ES6 nên bạn có thẻ coi như đây là một tính năng mới của Javascript.

**Cú pháp căn bản:**

Cú pháp căn bản nhất của arrow function như sau:

|  |
| --- |
| var functionName = (var1, var2) => {      // Nội dung function  }; |

**Ví dụ**: Viết arrow function in ra câu chào và so sánh với cách tạo function thông thường.

**Arrow function**:

|  |
| --- |
| var hello = (name, message) => {      console.log("Chào " + name + ", bạn là " + message);  };  hello('Cường', 'admin freetuts.net'); |

**Normal function**:

|  |
| --- |
| function hello(name, message)  {      console.log("Chào " + name + ", bạn là " + message);  }  hello('Cường', 'admin freetuts.net'); |

So sánh hai các trên thì rõ ràng cách thông thường sẽ đơn giản hơn rất nhiều, và cả hai đoạn code đều cho kết quả như sau:



**Nội dung là một câu lệnh đơn:**

Trường hợp trong thân của hàm chỉ có một lệnh duy nhất thì bạn có thể sử dụng theo ví dụ dưới đây.

|  |
| --- |
| var hello = (name, message) => console.log("Chào " + name + ", bạn là " + message); |

Nghĩa là bạn có thể bỏ đi cặp dấu {}, điều này tuân thủ theo nguyên tắc "*nếu bên thân cặp {} chỉ là một câu lệnh thì bạn có thể bỏ cặp {}*".

**Trường hợp một tham số:**

Trường hợp truyền vào chỉ một tham số thì bạn có thể bỏ cặp ().

|  |
| --- |
| var hello = message => {      console.log(message);  };  hello('Chào mừng bạn đến với freetuts.net'); |

**Trường hợp không có tham số:**

Trường hợp không có tham số truyền vào thì bạn sử dụng cặp () rỗng, xem ví dụ sau:

|  |
| --- |
| var hello = () => {      console.log('Chào mừng bạn đến với freetuts.net');  };  hello(); |

**2. Một số ví dụ arrow function trong ES6**

Bây giờ ta sẽ thực hành thông qua một số ví dụ để sau này các bạn không bị bỡ ngỡ khi gặp các đoạn code người ta viết trong thực tế.

**Ví dụ với hàm map:**

Ví dụ đầu tiên ta sử dụng kết hợp hàm map của array trong Javascript, hàm này giống như vòng lặp vậy, nó có một tham số truyền vào và đó là một **callback function**, hàm callback function này sẽ có hai tham số truyền vào đại diện cho **value** và **key**của mỗi phần tử trong mảng.

**Ví dụ**: Sử dụng hàm map để chuyển đổi các giá trị của các phần tử trong mảng sang chữ in hoa.

Sử dụng code thông thường:

|  |
| --- |
| var domain = ["freetuts.net", 'qa.freetuts.net', 'demo.freetuts.net'];  domain.map(function(val, key){      console.log(val.toUpperCase());  });  console.log(domain); |

Chạy lên kết quả sẽ như sau:



Nêu bạn thắc mắc tại sao danh sách domain sau khi xử lý (*tức là lệnh console.log(domain);*) lại không chuyển sang chữ in hoa thì ban phải xem lại đoạn code ở trên mình đã không lưu lại giá trị sau khi chuyển sang uppercase, vì vậy để giải quyết thì bạn chỉ việc lưu lại là được.

|  |
| --- |
| var domain = ["freetuts.net", 'qa.freetuts.net', 'demo.freetuts.net'];  domain.map(function(val, key){      console.log(val.toUpperCase());      // Lưu lại      domain[key] = val.toUpperCase();  });  console.log(domain); |

Chạy lên kết quả như hình sau:



Ok bây giờ ta sử dụng **Arrow Function** để viết.

|  |
| --- |
| var domain = ["freetuts.net", 'qa.freetuts.net', 'demo.freetuts.net'];  domain.map((val, key) => {      console.log(val.toUpperCase());  }); |

Quá đơn giản phải không các bạn :)

**Ví dụ với hàm setTimeout:**

Hàm [setTimeout](http://freetuts.net/settimeout-va-setinterval-trong-javascript-391.html) cũng có một **callback function** nên ta sẽ truyền vào callback đó một **Arrow Function**.

|  |
| --- |
| setTimeout(() => {      console.log('3 giây đã trôi qua');  }, 3000); |

Do arrow function không có tham số truyền vào nên mình chỉ để là ().

**3. Lỗi cú pháp với Arrow function**

Có một số lôi cú pháp khi sử dụng arrow function mà ta thường ít chú ý tới, nhin rất đơn giản nhưng đôi khi lại gây khó khăn cho những bạn mới học.

**Đóng arrow function**

Trường hợp bạn sử dụng arrow function bên trong một hàm hoặc sử dụng dạng một biến thì ban phải dùng cặp đóng mở để bao quanh lại.

|  |
| --- |
| console.log(typeof () => {}); // Cú pháp sai  console.log(typeof (() => {})); // Cú pháp đúng |

Trong ví dụ trên thì ví dụ đầu tiên sai vì arrow function được sử dụng này như một tham số, vì vậy bạn phải đặt nó bên trong cặp đóng mở như ở ví dụ 2. Trường hợp bạn không muốn đặt nó bên trong cặp đóng mở thì ban phải khai báo arrow function thành một biến như ví dụ dưới đây, tuy nhiên nhìn rất rườm rà.

|  |
| --- |
| var x = () => {}  console.log(typeof x); |

**Ràng buộc mũi tên**

Đúng với cái tên của nó là hàm mũi tên và mũi tên này rất khó chịu về cú pháp sử dụng, bạn phải đặt mũi tên cùng hàng với tên hàm.

|  |
| --- |
| const func1 = (x, y) // Sai  => {      return x + y;  };  const func2 = (x, y) => // Đúng  {      return x + y;  };  const func3 = (x, y) => { // OK      return x + y;  };  const func4 = (x, y) // Sai  => x + y;  const func5 = (x, y) => // Đúng  x + y; |

Nếu bạn muốn xuống hàng mà không bị lỗi thì phải sử dụng cú pháp sau:

|  |
| --- |
| const func6 = ( // Đúng      x,      y  ) => {      return x + y;  }; |

**4. Khắc phục nhược điểm với this trong closure function**

Nếu bạn đã từng đọc qua bài viết [hàm bind trong javascript](http://freetuts.net/hieu-hon-ve-ham-bind-trong-javascript-785.html) thì từ version ES5 trở về trước sẽ có nhược điểm với đối tượng this đó là phạm vi hoạt động, và trong ES5 có sử dụng hàm bind để khắc phục. Vấn đề này được khắc phục hoàn toàn trong ES6 bằng cách sử dụng hàm arrow function.

**Xét ví dụ sử dụng trong ES5 trở về trước**.

|  |
| --- |
| var blog = {      domain : "freetuts.net",      showWebsite : function (callbackFunction){          callbackFunction();      },      render : function(){          this.showWebsite(function (){             console.log(this.domain); // this chính là blog          }.bind(this)); // phải sử dụng hàm bind thì đối tượng this mới chính là blog      }  };  blog.render(); |

**Với ES6 thì viết như sau**:

|  |
| --- |
| var blog = {      domain : "freetuts.net",      showWebsite : function (callbackFunction){          callbackFunction();      },      render : function(){          this.showWebsite((() => {             console.log(this.domain); // this chính là blog          }));      }  };  blog.render(); |

**5. Lời kết**

ES6 quá thú vị phải không các bạn, mới chỉ bài viết thứ tư trong serise thôi mà có quá nhiều thứ bất ngờ và rất đáng để học hỏi.

Qua cách sử dụng **Arrow Function** thì cá nhân mình thấy vẫn chưa có điểm mạnh gì hơn so với các cách còn lại, vì vậy tùy vào sở thích của mỗi người mà chọn cách viết function cho phù hợp.

**Bài 05: Destructuring Assignments trong ES6**

Trong bài này chúng ta học một chức năng mới nữa trong ES6 đó là **Destructured Assignment**, chức năng này giống hàm list trong PHP vậy, nghia là nó sẽ phân các giá trị trong mảng vào các biến theo thứ tự được truyền vào.

Một ví dụ hàm list trong PHP:

|  |
| --- |
| $domain = array(      'freetuts.net',      'qa.freetuts.net'  );  // Gán hai phần tử vào hai biến $main và $sub  list($main, $sub) = $domain;  echo $main; // kết quả freetuts.net  echo $sub; // kết quả qa.freetuts.net |

Như vậy rõ ràng Javascript ES6 đang dần bắt kịp các ngôn ngữ khác. Tuy nhiên cá nhân mình lại thấy cách sử dụng này đôi khi lại khó kiểm soát code, vì vậy bài viết này chỉ mang tính chất giới thiệu chứ không khuyên bạn nên sử dụng. Và rõ ràng là chúng ta cũng nên bắt kịp công nghệ phải không nào :) Biết đâu CS7 ra đời sẽ có một cuộc cách mạng Javascript vĩ đại thì sao.

**1. Destructuring Assignments trong ES6**

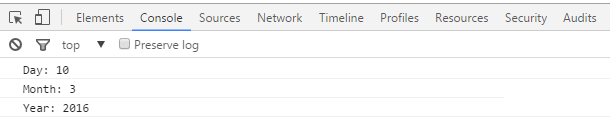
**Destructuring Assignments**đơn giản chỉ là cách tách các phần tử của Array hoặc Object thành nhiều biến chỉ bằng một đoạn code duy nhất.

**Sử dụng mảng:**

Bạn có thể sử dụng mảng để tách các phần tử thành các biến.

|  |
| --- |
| // Array  let date = [10, 03, 2016]  // Chuyển ba giá trị vào ba biến d, m, y  let [d, m, y] = date;  // In kết quả  console.log("Day: " + d);   // Day: 10  console.log("Month: " + m); // Month: 03  console.log("Year: " + y);  // Year : 2016 |

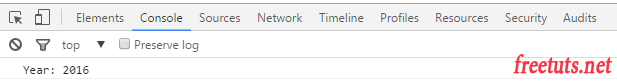
Chạy lên kết quả như sau:



Nếu bạn muốn lấy một phần tử thôi thì hãy bỏ trống các phần tử không muốn lấy.

|  |
| --- |
| // Array  let date = [10, 03, 2016]  // Chuyển ba giá trị vào ba biến y  let [, , y] = date;  // In kết quả  console.log("Year: " + y); |

Chạy lên kết quả như sau:

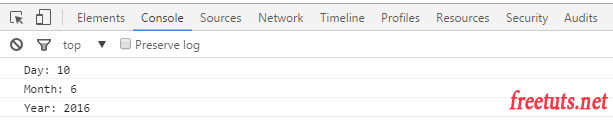


**Sử dụng object:**

Ngoài mảng ra thì bạn có thể tách biến từ object.

|  |
| --- |
| // Object  let date = {      day : 10,      month : 06,      year : 2016  }    // Chuyển ba giá trị vào ba biến  let {day : d, month : m, year : y} = date;    // In kết quả  console.log("Day: " + d);  console.log("Month: " + m);  console.log("Year: " + y); |

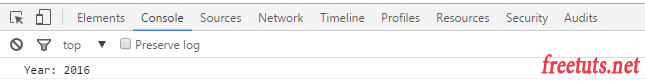
Kết quả:



Với trường hợp này thì bạn có thể lấy một phần tử bất kì chứ không bắt buộc theo thứ tự như bên mảng.

|  |
| --- |
| // Object  let date = {      day : 10,      month : 06,      year : 2016  }    // Chuyển ba giá trị vào ba biến y  let {      year : y  } = date;    // In kết quả  console.log("Year: " + y); |

Kết quả:



**2. Một sô ví dụ nâng cao khác**

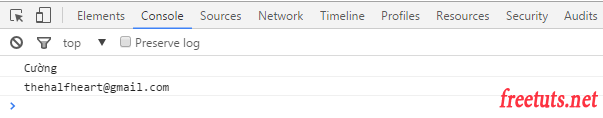
Bây giờ ta sẽ thực hành một số ví dụ nâng cao khác.

**Lấy giá trị theo key Object**

Bạn có thể lấy giá trị dựa vào tên key của Object.

|  |
| --- |
| var person = {name : "Cường", email : "thehalfheart@gmail.com"};    var {name : cuong\_name, email : cuong\_email} = person;    console.log(cuong\_name);    // Cường  console.log(cuong\_email);   // thehalfheart@gmail.com |

Kết quả:



Nhưng nếu bạn cố ý truy cập vào một key không tồn tại thì sẽ bị lỗi undefined.

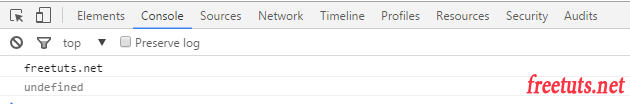
|  |
| --- |
| var {foo:bar} = {baz: 'ouch'};    console.log(bar); // Undefined |

**Lỗi undefined:**

Nếu phần tử không tồn tại thì sẽ bị undefined.

|  |
| --- |
| let domain = ['freetuts.net'];    let [main, sub] = domain;    // Xem kết quả  console.log(main);  console.log(sub); |

Kết quả do phần tử thứ hai không tồn tại nên biến sub sẽ có giá trị là undefined.

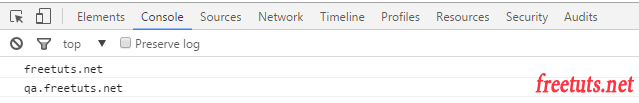


**Giá trị mặc đinh:**

Để tránh lỗi undefined thì bạn có thể gán giá trị mặc định cho nó.

|  |
| --- |
| let domain = ['freetuts.net'];    let [main, sub = 'qa.freetuts.net'] = domain;    // Xem kết quả  console.log(main);  console.log(sub); |

Kết quả sẽ không bị lỗi undefined.

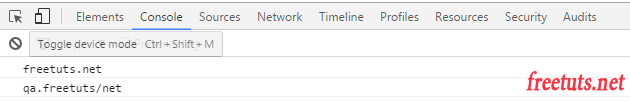


**Sử dụng function:**

Bạn có thể sử dụng function return về array hoặc object để gán vào thay vì gán một giá trị trực tiếp như vậy.

|  |
| --- |
| // Function  var domains = () => {      return [          'freetuts.net',          'qa.freetuts/net'      ];  };    // Chuyển đổi  let [main, sub] = domains();    // In kết quả  console.log(main);  console.log(sub); |

Kết quả:



**3. Lời kết**

Như vậy là mình đã giới thiệu xong sơ lược về cách sử dụng **Destructuring Assignments trong ES6**, đây là một phần khá phức tạp và đòi hỏi sự tập trung thì bạn mới có thể hiểu bài được.