CHAPTER 1

VALUES, TYPES, AND OPERATORS

Trong máy tính, những dữ liệu(data) được lưu trữ dưới dạng Bit (0 or 1)

**Values**:

Một máy tính hiện tại điển hình có khoảng 30 tỷ (30 billion) bits cho bộ nhớ làm việc (volatile data storage – working memory). Để có thể làm việc mà không bị mất một bit nào, nó phải chia chúng thành các phần đại diện cho thông tin (các chunks). Ở trong JavaScript thì mỗi **chunks được gọi là các values.** Tất các các values đều được tạo nên từ bits, đóng vai trò khác nhau. Mỗi một giá trị đều có một kiểu xác định vai trò của nó.Một vài giá trị (values) là số, một vài giá trị là đoạn văn bản, một vài giá trị là là hàm (functions) và hơn nữa.

Để tạo một giá trị, bạn chỉ cần gọi tên của nó. Điều này là thuận tiện. Bạn không cần phải thu thập vật liệu xây dựng cho các giá trị của mình hoặc trả tiền cho chúng. Bạn chỉ cần gọi một cái, và vù vù, bạn có nó. Tất nhiên, chúng không thực sự được tạo ra từ không khí loãng. Mọi giá trị phải được lưu trữ ở đâu đó và nếu bạn muốn sử dụng một lượng lớn chúng cùng một lúc, bạn có thể hết bộ nhớ. May mắn thay, đây là một vấn đề chỉ khi bạn cần tất cả chúng cùng một lúc. Ngay sau khi bạn không còn sử dụng một giá trị nữa, nó sẽ tiêu biến, để lại những bit của nó để được tái chế làm vật liệu xây dựng cho thế hệ giá trị tiếp theo.

Chương này giới thiệu các phần tử nguyên tử của các chương trình JavaScript, nghĩa là, các kiểu giá trị đơn giản và các toán tử có thể hoạt động trên các giá trị đó.

**Number:**

JavaScript sử dụng 64 bits để lưu trữ một giá trị số (single number value 2^64 giá trị)

Bộ nhớ máy tính từng nhỏ hơn nhiều và mọi người có xu hướng sử dụng các nhóm 8 hoặc 16 bit để biểu diễn các số của chúng. Rất dễ vô tình làm tràn những số nhỏ như vậy — để kết thúc bằng một số không phù hợp với số bit đã cho. Ngày nay, ngay cả những máy tính vừa với túi của bạn cũng có nhiều bộ nhớ, vì vậy bạn có thể thoải mái sử dụng các phần 64-bit và bạn chỉ cần lo lắng về việc tràn khi xử lý các con số thực sự lớn. Các bit đó cũng lưu trữ các số âm, vì vậy **một bit** chỉ ra dấu của số. Một vấn đề lớn hơn là các số thập phân cũng phải được biểu diễn. Để làm điều này, **một số** bit được sử dụng để lưu trữ vị trí của dấu thập phân. Dẫn đến số nguyên tối đa thực tế có thể lưu trữ lớn hơn 9 nghìn tỷ (2^53-1). Số phân số được viết bằng cách sử dụng dấu chấm.

Đối với các số rất lớn hoặc rất nhỏ, bạn cũng có thể sử dụng ký hiệu khoa học bằng cách thêm một e (đối với số mũ), theo sau là số mũ của số đó.

VD: 2.988e8 = 2.988 \* 10^8

Trong JS thì tính toán số nguyên (whole number) thì luôn cho kết quả CHÍNH XÁC, còn đối với các số thập phân thì cho kết quả TƯƠNG ĐỐI, do chỉ có 64 bits để lưu trữ mà nhiều số thập phân dài vô hạn vd PI

*Arithmetic:*

Operator: + add ,\* multiplication, / division, - subtraction, % modulo

Precedence: quyển ưu tiên

Có 3 giá trị đặc biệt trong JS :

1. Infinity: dương vô cùng
2. –infinity: Âm vô cùng
3. NaN: Not a Number vd: 0/0

**String**

Define: String là dạng dữ liệu cơ bản tiếp theo. Đại diện cho các dòng. Nội dung được viết ở bên trong dấu ngoặc (quotes)

Ex: ‘down on the sea’

“Lie on the ocean”

Bạn có thể sử dụng ngoặc đơn (Single quotes), ngoặc kép (Double quotes),backstick (``) để đánh dấu một String miễn là đầu cuối ngoặc phải cùng loại

bất cứ khi nào tìm thấy dấu gạch chéo ngược (\) bên trong văn bản được trích dẫn, nó chỉ ra rằng ký tự đứng sau nó có ý nghĩa đặc biệt. Điều này được gọi “*escaping the character*” =>> sử dụng **backslash escape character**

|  |  |  |
| --- | --- | --- |
| Code | Result | Description |
| \’ | ’ | Single quote |
| \” | ” | Double quote |
| \\ | \ | Backslash |
| \b | BackSpace | Xoá lùi |
| \f | Form Feed |  |
| \n | New line | Xuống dòng |
| \r | Carriage Return |  |
| \t | Horizontal tabulator | Tab (ngang) |
| \v | VerticalTabulator | Tab (dọc) |
| \ | None | Nhiều trình duyệt không hỗ trợ, nên bỏ đi |

Chuỗi cũng vậy, phải được mô hình hóa như một chuỗi các bit để có thể tồn tại bên trong máy tính. Cách JavaScript thực hiện điều này dựa trên tiêu chuẩn Unicode. Tiêu chuẩn này chỉ định một số cho hầu như mọi ký tự bạn cần, bao gồm các ký tự từ tiếng Hy Lạp, Ả Rập, Nhật Bản, Armenia, v.v. Nếu chúng ta có một số cho mọi ký tự, một chuỗi có thể được mô tả bằng một chuỗi số.

JS sử dụng 16 bits cho mỗi phần tử của chuỗi => có thể mô tả 2^16 ký tự khác nhau.

Chuỗi thì không thể nhân (\*) , chia (/), hay trừ (-), nhưng có thể nối lại bằng dấu (+), được gọi là *concatenates:*

Vd: concatenates = “con” + ‘cate’ + “nates”

Các chuỗi được viết bằng dấu ngoặc đơn hoặc dấu ngoặc kép thì giống nhau. Còn chuỗi được viết bằng dấu Backtick-quote thường gọi là các ký tự mẫu (template literals), dùng để thực hiện các thao tác thủ thuật, nhúng các giá trị khác.

Vd: `half of 100 is ${100 / 2}` =>> half of 100 is 50

**`${arguments}`**

**Khi viết biểu thức bên trong ${} của một *template literals* thì nó sẽ được tính toán và trả về đúng vị trí đấy**.

**Unary Operator (Toán tử 1 ngôi)**

Không phải tất cả các toán tử đều ở dạng ký hiệu. Một vài toán tử được viết dưới dạng từ ví dụ như ***typeof***  trả về tên của dạng giá trị mà bạn đang có:

|  |
| --- |
|  |

Toán tử 1 ngôi: ++ and –

Có 2 khái niệm cần nhớ: *prefix (tiền tố)* and *postfix(Hậu tố)*. Cách sử dụng khác nhau.

Vd: var a = 5

a++ hoặc ++a

console.log(a) // đều cho ra kết quả là 6 nhưng thực hiện khác nhau

*prefix (Tiền tố)*

b = ++a;

console.log(b) =>> cho ra b = a + 1 = 6

Trả về giá trị của a sau khi đã thực hiện phép tính.

*Postfix (Hậu tố)*

c = a++;

console.log( c) =>> cho ra c = a = 5

Trả về giá trị ban đầu của a

Toán tử yêu cầu 2 giá trị thì là toán tử 2 ngôi (Binary Opartors) và toán tử yêu cầu 1 giá trị là toán tử 1 ngôu (Unary Operators).

***Boolean Values* (Giá trị boolean)**

Giá trị Boolean có 2 loại với những mục đích khác nhau (True - False), (On - Off), (0-1)

Những cách để có được giá trị Boolean : **comparition**

|  |
| --- |
| Note: So sánh String thì sẽ so sánh các ký tự tù trái qua phải tại cùng 1 một vị trí |

Chỉ có một giá trị trong JS không bằng chính nó đó chính là : NaN (Not a Number)

|  |
| --- |
|  |

Bất cứ giá trị nào trong Javascript khi chuyển đổi sang kiểu dữ liệu boolean mà có giá trị **true** thì ta gọi giá trị đó là **truthy**.

Các giá trị 1, ['BMW'], { name: 'Miu' } và 'hi' được đề cập trong ví dụ dưới đây là **truthy** vì khi chuyển sang **boolean** ta nhận được giá trị **true**.

|  |
| --- |
|  |

!! là gì? Đơn giản thôi. Toán tử ! là toán tử not (phủ định) nên !! là 2 lần phủ định, mà 2 lần phủ định lại trở thành “khẳng định”. Trong Javascript thì đây là một “tip” để convert (chuyển đổi) mọi kiểu dữ liệu khác sang Boolean.

Thêm !! phía trước các giá trị truthy sẽ luôn trả về true.

|  |
| --- |
|  |

**LOGICAL OPERATORS (Toán tử Logic)**

Toán tử dùng để áp dụng cho các giá trị boolean. Có 3 loại toán tử logic: and, not, or.

|  |  |  |
| --- | --- | --- |
| Logical operators | Symbol | Example |
| And | && |  |
| Or | || |  |
| Not | ! | !true = false |

Toán tử 3 ngôi (*ternary operator* or *conditional operator*):

Syntax:

|  |
| --- |
| Condition ? value\_true : value\_Flase |

|  |
| --- |
|  |

**Empty Values (Giá trị trống)**

Có 2 giá trị đặc biệt mà không mang thông tin là *null* và *undefine*

Chúng đều là các giá trị nhưng không mang thông tin nào cả. Có thể sử dụng cả 2 thay thế lẫn nhau.

Vd:

Var age;

Console.log(age) =>> undefine

Var isNull = null

**Automatic Type Conversion:**

|  |
| --- |
|  |

Khi một toán tử được áp dụng cho 2 giá trị không cùng loại thì JS sẽ mặc định chuyển giá trị về một loại giá trị cần thiết, sử dụng các quy tắc. nó được gọi là: “*type coercion*”

Khi so sánh 2 giá trị cùng loại bằng ==, rất dễ để nhận thấy rằng bạn sẽ nhận giá trị TRUE nếu cả 2 có cùng giá trị ngoại trừ trường hơp NaN. Nhưng trong trường hợp khác loại, JS sử dụng các phương thức phức tạp để chuyển đổi giá trị của loại này về giá trị của loại khác. Tuy nhiên khi cả Null và Undefined đều xuất hiện ở 2 vế thì sẽ trả về giá trị TRUE luôn

|  |
| --- |
|  |

**Bảng note**

|  |  |
| --- | --- |
| List of values | Convert to Boolean Type |
| 0 | False |
| ‘’ – “” (Empty String) | False |
| undefined | False |
| NaN (Not a Number) | False |
| null | False |
| Other values | True |

Hành vi này rất có ích vì khi bạn muốn kiểm tra xem một giá trị có giá trị thực thay vì *null* hay *undefined* thì có thể so sánh nó với *null* qua toán tử == hay !=.

Nhưng điều gì sẽ xảy ra nếu bạn muốn kiểm tra xem một cái gì đó đề cập đến giá trị chính xác là false? Các biểu thức như “0 == false” và ‘ "" == false ‘ cũng đúng vì chuyển đổi kiểu tự động. Khi bạn không muốn bất kỳ chuyển đổi kiểu này xảy ra, có hai toán tử bổ sung: === và! ==. Phép thử đầu tiên kiểm tra xem một giá trị có chính xác bằng giá trị kia hay không và phép thử thứ hai kiểm tra xem giá trị đó có chính xác bằng nhau hay không. Vì vậy, "" === false là false như mong đợi.

Chúng ta nên dùng toán tử so sánh là ‘===’ và ‘!==’ để so sánh tránh sự chuyển đổi kiểu không mong muốn. Còn nếu biết rõ loại của cả 2 vế giống nhau thì có thể sử dụng “==” hoặc ‘!=’

**Short-Circuiting Of Logicial Operators**

Các toán tử logic && và || xử lý các giá trị của các kiểu khác nhau theo một cách đặc biệt. Họ sẽ chuyển đổi giá trị ở phía bên trái của họ sang kiểu Boolean để quyết định phải làm gì, nhưng tùy thuộc vào toán tử và kết quả của chuyển đổi đó, họ sẽ trả về giá trị bên trái ban đầu hoặc giá trị bên phải.

Cái || chẳng hạn, toán tử sẽ trả về giá trị bên trái khi giá trị đó có thể được chuyển thành true và ngược lại sẽ trả về giá trị ở bên phải. Điều này có hiệu quả mong đợi khi các giá trị là Boolean và thực hiện điều gì đó tương tự đối với các giá trị của các loại khác.

|  |
| --- |
|  |

Chúng ta có thể sử dụng chức năng này như một cách để lấy lại giá trị mặc định. Nếu bạn có một giá trị có thể trống, bạn có thể đặt || sau nó với một giá trị thay thế. Nếu giá trị ban đầu có thể được chuyển đổi thành false, bạn sẽ nhận được giá trị thay thế. Các quy tắc để chuyển đổi chuỗi và số thành giá trị Boolean nêu rõ rằng 0, NaN và chuỗi trống ("") được tính là FALSE, trong khi tất cả các giá trị khác được tính là TRUE. Vì vậy, 0 || -1 tạo ra -1 và "" || "!?" kết quả "!?".

Toán tử && thì hoạt động tương tự nhưng ngược lại. X && Y : nếu X là False thì sẽ trả về X còn không trả về giá trị của Y.

Một thuộc tính quan trọng của những toán tử này là phần vế phải của toán tử được đánh giá chỉ khi cần thiết. Trong trường hợp True || X, bất kể X là gì — ngay cả khi đó là một phần của chương trình thực hiện điều gì đó khủng khiếp — kết quả sẽ là true và X không bao giờ được đánh giá. Tương tự với FALSE && X thì X cũng sẽ được bỏ qua. Nó được gọi là: “short-circuit evaluation” (Đánh giá ngắn mạch).

Trong toán tử logic && nếu vế đầu tiên của biểu thức trả về giá trị **false** thì biểu thức sẽ nhận luôn giá trị này (1 trong 6 giá trị được nêu ra ở mục note) còn nếu không nó sẽ nhận được giá trị cuối cùng

Ví dụ

|  |
| --- |
| Var x = ‘a’ && “B” && 5 =>> x = 5 |
| Var y = null && ‘1’ =>> y = null  Var z = ‘’ && null =>> z = ‘’ |

Ngược lại với toán tử && thì dùng toán tử ||, chỉ cần tìm được giá trị khác 6 giá trị note trên thì sẽ trả về giá trị đó luôn.

Toán tử điều kiện(3 ngôi) hoạt động tương tự, quan tâm đến giá trị đầu tiên hơn.

**SUMMARY**

Chúng ta có 4 loại giá trị trong JS: numbers, Strings, Boolean, và undefined.

Các giá trị như vậy được tạo bằng cách nhập tên của chúng (true, null) hoặc giá trị (13, "abc"). Bạn có thể kết hợp và biến đổi các giá trị bằng các toán tử. Chúng tôi đã thấy các toán tử nhị phân cho số học (+, -, \*, /, và%), nối chuỗi (+), so sánh (==,! =, ===,! ==, <,>, <=,> = ), và logic (&&, ||), cũng như một số toán tử một bậc (- để phủ định một số,! để phủ định một cách logic và typeof để tìm kiểu của một giá trị) và một toán tử bậc ba (? :) để chọn một trong hai giá trị dựa trên giá trị thứ ba.

Điều này cung cấp cho bạn đủ thông tin để sử dụng JavaScript như một máy tính bỏ túi nhưng không nhiều hơn thế. Chương tiếp theo sẽ bắt đầu liên kết các biểu thức này với nhau thành các chương trình cơ bản.

🡺 **check the type of the varialbe : console.log(typeof var);**