FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

# JAVASCRIPT NÂNG CAO

## Lập trình hướng đối tượng trong Javascript

THỰC HỌC – THỰC NGHIỆP

http://www.poly.edu.vn

# Mục tiêu

- Giải thích được khái niệm về hướng đối tượng, sử dụng đối tượng như công cụ trong công việc.
- Nắm được các cách triển khai đối tượng trong javascript
- Triển khai lớp kế thừa trong javascript

# NỘI DUNG

- 📖Khái niệm hướng đối tượng trong javascript
- 📖Sử dụng được Object literals
- 📖Sử dụng được Object constructor functions
- 📖Giải thích được Prototyping inheritance

# NHẮC LẠI BÀI CŨ

- 📖Ôn tập lại các hàm xử lý chuỗi
- 📖Giải thích được khái niệm và các cách sử dụng regex
- 📖Giải thích được các khái niệm Pattern/Modifiers trong regex
- 📖Sử dụng được các cú pháp regex cho pattern
- 📖Giải thích được xử lý ngoại lệ, khái niệm try – catch – finally


# PHẦN 1:
# KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG

# KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG

- Javascript được thiết kế dựa trên mô hình đối tượng đơn giản (object-based paradigm).
  - Một đối tượng là một tập hợp các thuộc tính và một thuộc tính là một liên kết giữa một tên (hoặc khóa) và một giá trị (key-value).
  - Giá trị của một thuộc tính có thể là một hàm, trong trường hợp đó, thuộc tính được gọi là một phương thức.
  - Ngoài các đối tượng được xác định trước trong trình duyệt, bạn có thể xác định các đối tượng của riêng mình.

Nguồn: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

# KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG

- Các đối tượng trong Javascript, giống như trong nhiều ngôn ngữ lập trình khác, có thể được so sánh với các đối tượng trong cuộc sống thực.
- Khái niệm về các đối tượng trong JavaScript có thể được hiểu với cuộc sống thực, các đối tượng hữu hình.
- Trong JavaScript, một đối tượng là một thực thể độc lập, với các thuộc tính và loại.
- Ví dụ: mô tả một chiếc cốc chẳng hạn, chiếc cốc là một đối tượng, với các thuộc tính. Một chiếc cốc có màu sắc, kiểu dáng, trọng lượng, chất liệu, giá tiền, v.v ... Tương tự, các đối tượng Javascript có thể có các thuộc tính, các đặc điểm của riêng mình.

Nguồn: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

# KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG

- Sử dụng đối tượng trong javascript không có gì xa lạ, vì về cơ bản mọi thứ trong javascript đều là đối tượng.
- Ví dụ:
  - Chuỗi
  - Mảng
  - Số
  - ...

```
> var str2Arr = "fpt poly javascript nâng cao".split(" ");
  var reverseArr = [1, 2, 5, 9, 12, 20].reverse();
  console.log(str2Arr); console.log(reverseArr);
  ▶ (5) ["fpt", "poly", "javascript", "nâng", "cao"]
  ▶ (6) [20, 12, 9, 5, 2, 1]
```

- Do đó khi sử dụng các api trong các bài học trước chúng ta gọi các hàm là phương thức.

# KHÁI NIỆM HƯỚNG ĐỐI TƯỢNG

- Các cách tự tạo đối tượng trong javascript:
  - Object literals
  - Object constructor functions

# PHẦN 2:
# SỬ DỤNG OBJECT LITERALS

# SỬ DỤNG OBJECT LITERALS

- ***Object literals*** được biểu diễn bằng dấu phẩy ngăn cách giữa các cặp *name-value* nằm trong ngoặc nhọn.
- Thuộc tính *name* thuộc kiểu dữ liệu dạng chuỗi (strings) và giá trị là một trong số bất kỳ kiểu dữ liệu nào của javascript như mảng (arrays), chuỗi (strings), số (number) hay hàm (function)...
- Sử dụng object literals chúng ta có thể gom các giá trị (properties) và phương thức (method) vào cùng nhau giúp code sạch và dễ đọc hơn.

# SỬ DỤNG OBJECT LITERALS

- Ví dụ: Tạo object mới

```
var alex = {
    idCard: 16323891,
    name: "Alex Smith",
    age: 45,
    isMarried: true,
    childs: [
        {
            name: "James Smith",
            age: 13
        },
        {
            name: "Sarah Smith",
            age: 7
        }
    ],
    buyHouse: function(){
        console.log('already own a house in Pennsylvania');
    }
}
```

# SỬ DỤNG OBJECT LITERALS

- **Hiển thị giá trị thuộc tính/gọi phương thức**

```
> // lấy giá trị của thuộc tính
  console.log(alex.name); console.log(alex.age);
  // thực thi phương thức của đối tượng alex
  alex.buyHouse();
  Alex Smith
  45
  already own a house in Pennsylvania
```

- **Thay đổi giá trị của thuộc tính**

```
> // sửa giá trị của thuộc tính
  console.log(alex.age);
  alex.age = 48;
  console.log(alex.age);
  // thêm giá trị cho thuộc tính dạng mảng
  console.log(alex.childs);
  var newChild = {name: "Jim Smith", age: 1};
  alex.childs.push(newChild);
  console.log(alex.childs);
  45
  48
  ▼ (2) [{…}, {…}] i
    ▶ 0: {name: "James Smith", age: 13}
    ▶ 1: {name: "Sarah Smith", age: 7}
    ▶ 2: {name: "Jim Smith", age: 1}
      length: 3
    ▶ __proto__: Array(0)
  ▼ (3) [{…}, {…}, {…}] i
    ▶ 0: {name: "James Smith", age: 13}
    ▶ 1: {name: "Sarah Smith", age: 7}
    ▶ 2: {name: "Jim Smith", age: 1}
      length: 3
    ▶ __proto__: Array(0)
```

# SỬ DỤNG OBJECT LITERALS

- Bên cạnh việc tạo object bằng ký tự {} thì chúng ta có thể tạo bằng câu lệnh new Object().

```
var rex = new Object();
rex.name = 'Rex Kumar';
rex.age = 23;
rex.gender = "male";
rex.goShopping = function(){
    console.log(this.name + ' is going to Super market');
}
console.log(rex);
rex.goShopping();
```

OUTPUT

```
▼ Object i
    age: 23
    gender: "male"
  ▶ goShopping: f ()
    name: "Rex Kumar"
  ▶ __proto__: Object
Rex Kumar is going to Super market
```

# SỬ DỤNG OBJECT LITERALS

- Lưu ý với cách sử dụng object literals thì lợi thế là chúng ta có thể tạo ra object tùy ý và ở bất kỳ đâu trong dự án, tuy nhiên cách sử dụng này cũng có những hạn chế đó là:
  - Không hỗ trợ thực thể hóa đối tượng (instantiation)
  - Không có tính chất kế thừa (inheritance)

# SỬ DỤNG OBJECT LITERALS

- Luyện tập: Tạo một đối tượng sinh viên bằng phương pháp sử dụng object literals, đối tượng bao gồm các thuộc tính/phương thức sau
  - ma_sinh_vien
  - ho_va_ten
  - lop
  - ngay_nhap_hoc
  - diem_trung_binh
  - tinhHocLuc()
  - thoiGianRaTruong()

# SỬ DỤNG OBJECT LITERALS

- Phương thức tinhHocLuc() trả về học lực của sinh viên theo thang điểm:
  - < 5: Học lực Yếu
  - Từ 5 đến < 7 điểm: Học lực Trung bình
  - Từ 7 đến 8: Học lực Khá
  - Từ 8 đến 10: Học lực Giỏi
- Phương thức thoiGianRaTruong() trả về số tháng còn lại theo dự kiến tổng thời gian học là 28 tháng. Ví dụ nhập học ngày 04/01/2019 thì dự kiến sẽ còn 17 tháng nữa sẽ ra trường.

# PHẦN 3:
# OBJECT CONSTRUCTOR FUNCTIONS

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- ***Object constructor functions*** hay tạo khuôn mẫu một đối tượng bằng cách sử dụng hàm (khởi tạo). Đây là một trong những cách thông dụng nhất để tạo ra một object, cách này có các ưu điểm sau:
  - Có thể thực thể hóa các đối tượng (instantiation)
  - Có thể được kế thừa hoặc cho phép kế thừa (inherited)

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

## Tạo đối tượng sử dụng Object constructor functions

```
function Animal(name, age, gender){
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.run = function(){
        console.log(this.name + " is running!");
    }
}

var duck = new Animal('Donald', 2, 'male');
duck.hairColor = 'yellow';
console.log(duck);
duck.run();
```

OUTPUT

```
▼Animal {name: "Donald", age: 2, gender: "male", hairColor: "yellow", run: f} i
    age: 2
    gender: "male"
    hairColor: "yellow"
    name: "Donald"
  ▶ run: f ()
  ▶ __proto__: Object
Donald is running!
```

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- Cách tạo đối tượng bằng Object constructor functions khá giống với các ngôn ngữ backend như Java, C#, PHP,... khác ở chỗ sử dụng hàm thay cho class.
- Từ khóa this – trỏ đến thực thể (instance) vừa được tạo ra bởi cú pháp new Animal()
- Để thực hiện kế thừa giữa các lớp đối tượng với nhau, với phương pháp tạo đối tượng này chúng ta cần sử dụng đến phương thức call().
- Ý nghĩa của hàm call(): sử dụng để có thể viết một phương thức một lần duy nhất và sau đó thừa kế nó trong một đối tương khác mà không phải viết lại phương thức đó cho đối tượng mới.

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- Ví dụ sử dụng hàm call() để thực hiện kế thừa giữa 2 đối tượng

```
function Animal(name, age, gender){
    this.name = name;
    this.age = age;
    this.gender = gender;
    this.run = function(){
        console.log(this.name + " is running!");
    }
}

function Cat(name, age, gender, hairColor){
    Animal.call(this, name, age, gender);
    this.hairColor = hairColor;
}

var doraemon = new Cat('Doraemon', 27, 'male', 'green');
console.log(doraemon);
doraemon.run();
```

OUTPUT

```
▼ Cat {name: "Doraemon", age: 27, gender: "male", hairColor: "green", run: f} i
    age: 27
    gender: "male"
    hairColor: "green"
    name: "Doraemon"
  ▶ run: f ()
  ▶ __proto__: Object
Doraemon is running!
```

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- Luyện tập: tạo lớp VanDongVien bằng phương pháp Object constructor functions gồm các thuộc tính/phương thức sau:
  - ma_van_dong_vien
  - ho_va_ten
  - ngay_sinh
  - giai_thuong
  - kiemTraDieuKien()
  - themGiaiThuong()
  - danhSachGiaiThuong()

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- Phương thức **kiemTraDieuKien()** trả về true/false, nhận 2 tham số là:
  - Chuỗi ngày tháng năm (ngày tổ chức giải đấu tiếp theo)
  - Số tuổi đủ điều kiện tham gia thi đấu
- Yêu cầu 1: viết code cho hàm để kiểm tra xem với ngày tháng năm sinh của mình, vận động viên có đủ điều kiện tham gia thi đấu hay không?

# SỬ DỤNG OBJECT CONSTRUCTOR FUNCTIONS

- Thuộc tính **giai_thuong** là dạng mảng chứa các phần tử dạng chuỗi để lưu lại các giải thưởng của vận động viên đạt được.
- Phương thức **themGiaiThuong()** trả về số lượng giải thưởng của vận động viên, nhận tham số là:
  - Chuỗi là tên giải thưởng của vận động viên nhận được
- Yêu cầu 2: viết code cho phương thức **themGiaiThuong()** để thêm phần tử cho thuộc tính **giai_thuong**.
- Yêu cầu 3: viết code cho phương thức **danhSachGiaiThuong()** để hiển thị danh sách các giải thưởng mà vận động viên đã đạt được


# PHẦN 4:
# SỬ DỤNG PROTOTYPING INHERITANCE

# KHÁI NIỆM PROTOTYPE

## Ví dụ 1:

Code javascript

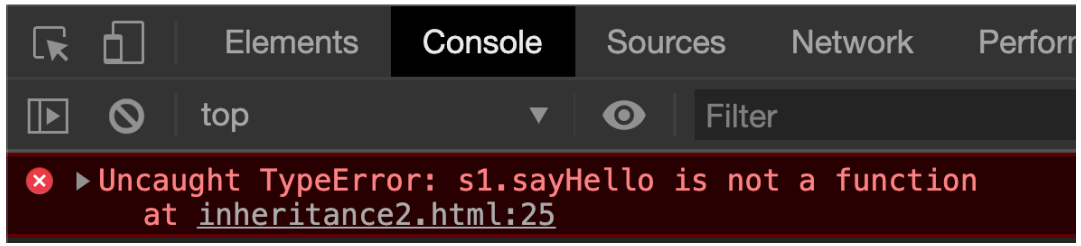
```
function Student(name, age) {
    this.name = name;
    this.age = age;
}

const s1 = new Student('Bad Cuder', 20);

Student.sayHello = function () {
    console.log('Hello');
}

s1.sayHello();
// Lỗi, vì s1 ko có hàm sayHello, hàm này chỉ thuộc Student thôi.
```

Output lỗi



# KHÁI NIỆM PROTOTYPE

- Từ ví dụ trên ta có thể nhận thấy được khi thêm hàm sayHello() vào trực tiếp Student thì chỉ có thể được gọi trực tiếp từ Student chứ không thể truy cập được từ instance s1 được tạo bởi new Student().
- Do đó chúng ta cần phải có giải pháp để có thể sửa nội dung của Student (ví dụ thêm phương thức mới) mà các instance (ví dụ như s1) được tạo ra từ đó cũng có thể sử dụng được .

# KHÁI NIỆM PROTOTYPE

## ❑ Hãy xem ví dụ 2:

Code javascript

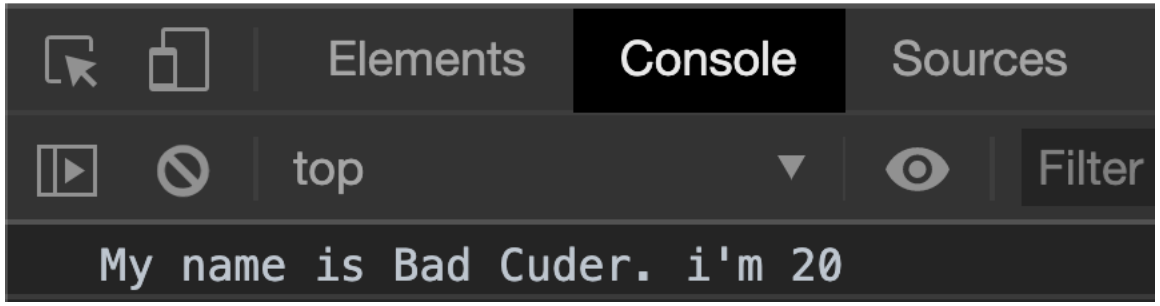
```
function Student(name, age) {
    this.name = name;
    this.age = age;
}

const s1 = new Student('Bad Cuder', 20);

Student.prototype.sayHello = function () {
    console.log('My name is ' + this.name
                + ". i'm " + this.age);
}

s1.sayHello();
```

Output



# KHÁI NIỆM PROTOTYPE

- Ta có thể nhận thấy, nếu hàm sayHello() được tạo ra khi ta gọi từ prototype thì các instance (s1) có thể gọi được hàm đó.
- Vậy ta có thể phát biểu như sau: prototype là nơi định nghĩa ra các thuộc tính, phương thức (hàm) cho khuôn mẫu của 1 đối tượng, các thực thể (instance) được tạo ra từ đối tượng đó được quyền sử dụng các thuộc tính, phương thức được tạo ra bằng prototype – một cách tạo constructor rất thú vị của javascript.

# SỬ DỤNG PROTOTYPING INHERITANCE

- Cho ví dụ 3:

```
var people = {
    name: "Dang Quang Duc",
    age: 2,
    showInfo: function(){
        return "Chau ten la " + this.name + " nam nay chau " + this.age;
    }
}

var student = {
    name: "Tran Trang Anh",
    age: 4,
    showInfo: function(){
        return "Chau ten la " + this.name + " nam nay chau " + this.age;
    }
}
```

# SỬ DỤNG PROTOTYPING INHERITANCE

- Trong ví du 3 chúng ta có thể quan sát được cả 2 đối tượng đều sử dụng chung phương thức showInfo(), như vậy nếu các đối tượng được tạo ra từ phương pháp Object Literals thì các thuộc tính hay phương thức sẽ cần phải viết lại khá nhiều, gây phức tạp khi tiến hành maintainance sau này.
- Trong javascript có cung cấp thuộc tính __proto__ sẽ hỗ trợ giải quyết vấn đề rút gọn code (có thể coi là một dạng kế thừa) khi sử dụng object.
  - Thuộc tính __proto__ thực ra là 1 *getter function*, *getter function* này được định nghĩa trên **native built-in Object** của JavaScript. Có thể coi như 1 cách trình duyệt  để bạn truy cập vào Prototype của Object

# SỬ DỤNG PROTOTYPING INHERITANCE

## Cho ví dụ 4:

Code javascript

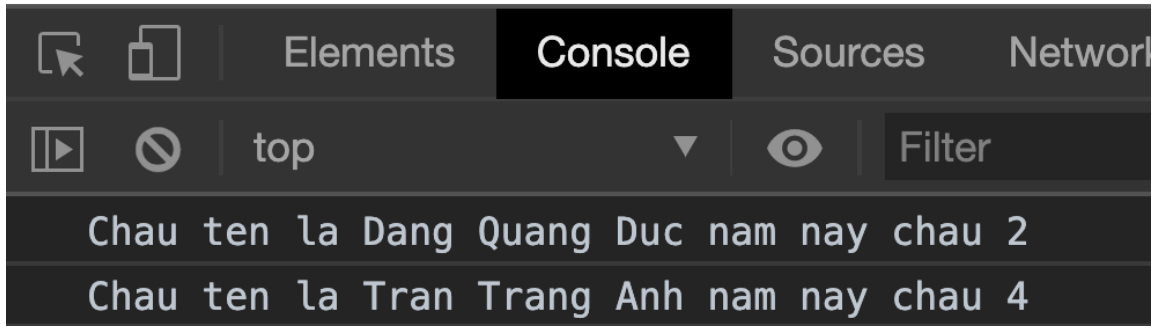
```
var PeopleProto = {
    name: "Dang Quang Duc",
    age: 2,
    showInfo: function(){
        return "Chau ten la " + this.name
            + " nam nay chau " + this.age;
    }
}

var quangDuc = {
    __proto__: PeopleProto
}

var trangAnh = {
    __proto__: PeopleProto
}
trangAnh.name = "Tran Trang Anh";
trangAnh.age = 4;

console.log(quangDuc.showInfo());
console.log(trangAnh.showInfo());
```

Output



# SỬ DỤNG PROTOTYPING INHERITANCE

- Như ví dụ 4 ta có thể xây dựng ra 1 khung sườn (base) cho một đối tượng (gọi là A), sau đó sử dụng thuộc tính __proto__ để đưa đối tượng đó trở thành khuôn mẫu cho các đối tượng khác (B và C), các đối tượng kế thừa (B, C) đối tượng gốc (A) kia có thể tái sử dụng lại các phương thức/thuộc tính được định nghĩa từ đối tượng A. Có thể coi đây là 1 cách để tạo constructor cho các đối tượng trong javascript.

# TỔNG KẾT

- ☑ Khái niệm đối tượng trong javascript
- ☑ Tạo đối tượng bằng Object Literals
- ☑ Tạo đối tượng bằng Object constructor functions
- ☑ Khái niệm prototype
- ☑ Cách sử dụng Prototyping Inheritance


TIN HỌC VĂN PHÒNG QUỐC TẾ THEO CHUẨN ICDL
Syllabus 5 - Office 2007
LUẬT KINH TẾ
LUẬT KINH TẾ
thank you!