OOP

Object Oriented Programming

Why OOP?

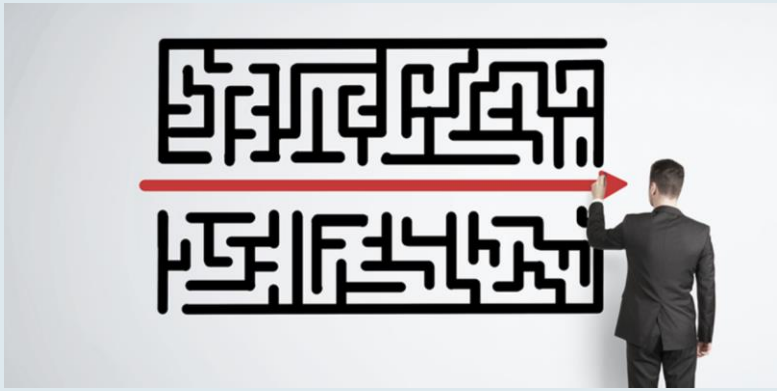
What is our motivation to learn Object Oriented Programming?

Why OOP?

Simplicity

One important way to deal with complexity is to reduce it to something simpler.

Divide and conquer principle: Divide the original problem into separate sub-problems that can be solved individually.



Why OOP?

Maintaibility

We must also be able to maintain and extend the solution in an evolving world.

Why OOP?

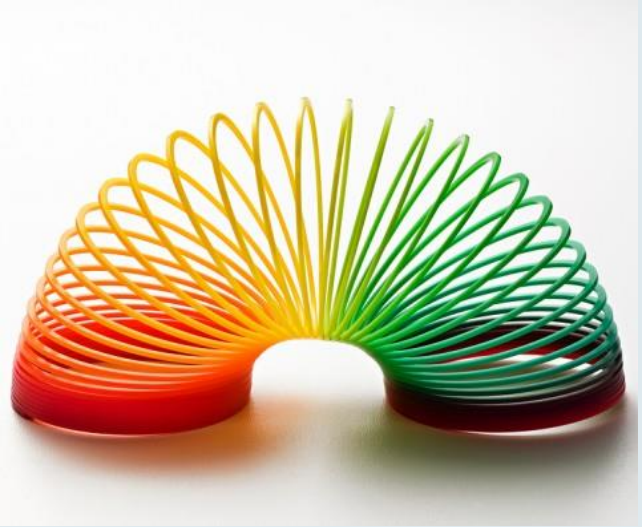
Reuseability

When you divide problems into sub-problems, you will sometimes see similar sub-problems. Then it would be nicer to reuse a similar solution.

Why OOP?

Flexibility

Our software should be easily open for changes and extensions without requiring to change the entire code base



A Model to Create Objects

Let's say we have a small farm with some animals.

How can we describe it?

We could create an object for each animal:

```
let animal1 = {
  name: "Fufu",
  type: "horse",
  age: 7
}

let animal2 = {
  name: "Mimi",
  type: "sheep",
  age: 3
}
```

But that is a lot of work!

OOP with ES6

EcmaScript 6

OOP

Modeling abstract ideas into known domains

For example – a car is an abstract concept.

OOP helps us translate it into the world we know:

- Objects
- Primitive types
- Functions

OOP

Attributes (fixed)
- Color: Blue
- Brand
- Wheels: 4
- Engine: 1.2L, 80HP

State
- Lights: ON / OFF
- Engine: ON / OFF
- Speed: 0-210 mph

Functionality
- Drive
- Stop
- Turn Left
- Turn Right

Class

A Model

It is important to note that the car model is **not** describing a specific car, but a way to represent many different cars.

We call this model a class.

```
class Car {
    constructor(color, brand){
        this.color = color;
        this.brand = brand;
    }
}
```

Classes and Constructors

ES6 class

We use the **class** keyword to declare a class

Class name starts with a capital letter

```
class Car {
   constructor(color, brand){
      this.color = color;
      this.brand = brand;
   }
}
```

We declare a <u>constructor</u> with a function named constructor (saved word).

In the <u>constructor</u> we define the class attributes.

The class is surrounded with curly braces

Meet the constructor

The **new** keyword calls a function: **constructor**

Calling the constructor creates an instance of an object

- The name of the class (Car) is the name of the object "type" so the object `swift` is a `Car` instance.
- Class name should always start with a capital letter.

```
class Car {
   constructor(color, brand){
      this.color = color;
      this.brand = brand;
   }
}
let swift = new Car("green", "Suzuki");
```

Here we invoke the constructor function and create a Car instance.

Meet the constructor

Instance creation flow:

```
class Car {

   constructor(color, brand){
      this.color = color;
      this.brand = brand;
   }

}


let swift = new Car("green", "Suzuki");
let civic = new Car("black", "Honda");
```

Exercise:
Debug this code to understand the instance creation flow. Work in pairs and explain to each other the code flow.

Meet the constructor

Instance creation flow:

```
class Car {

   constructor(color, brand){
      this.color = color;
      this.brand = brand;
   }

}


let swift = new Car("green", "Suzuki");
Let's debug it.
```

2. The constructor function is invoked

3. The parameters that we passed ("green", "suzuki") are assigned to the new instance

1. Creating a new instance with the new keyword

Constructor Parameters

Instance creation flow:

Here we define the parameters that will be use to initialize every instance

```
class Car {

   constructor(color, brand){
      this.color = color;
      this.brand = brand;
   }

}
let swift = new Car("green", "Suzuki");
or
let carColor = "green";
let carBrand = "Suzuki";
let swift = new Car(carColor, carBrand);
```

Using the parameters that we defined and assign them to the new instance

Passing the arguments to the constructor, the actual values of the specific instance

Terminology Time!

```
class Animal {
  constructor(name, type, age){
    this.name = name;
    this.type = type;
    this.age = age;
  }
}

let animal1 = new Animal("Fufu", "horse", 7);
let animal2 = new Animal("Mimi", "sheep", 3);
```

Class:
Defines the general attributes and behavior

Instance

Instance

Instance:
Specific implementation of the class. Each instance may have different attributes.

Instance

```
class Animal {
  constructor(name, type, age){
    this.name = name;
    this.type = type;
    this.age = age;
  }
}
let animal1 = new Animal("Fufu", "horse", 7);
let animal2 = new Animal("Mimi", "sheep", 3);
```

So animal1 is an object {} with fields:

```
name = "Fufu"
type = "horse"
age = 7
```

Almost as if we defined:

```
let animal1 = {
  name: "Fufu",
  type: "horse",
  age: 7
};
```

If we print it in The console it looks like this:

```
> animal1
<· ▼Animal {name: "Fufu", type: "horse", age: 7}
    age: 7
    name: "Fufu"
    type: "horse"
  ▶ __proto__: Object
```

Exercise – Car instances

Create Instances

```
class Car {
    constructor(model, brand){
        this.model = model;
        this.brand = brand;
    }
}
```

Your task: create 3 different car instances.

Example of cars

We can create from the same class different cars

Car
Color: blue
Brand: Honda
Speed: 0
drive(), stop()

```
let car1 = new Car("blue", "Honda");
```

Car
Color: gray
Brand: Suzuki
Speed: 90
drive(), stop()

```
let car2 = new Car("gray", "Suzuki");
```

Car
Color: black
Brand: Ford
Speed: 110
drive(), stop()

```
let car3 = new Car("black", "Ford");
```

Car
Color: white
Brand: Toyota
Speed: 0
drive(), stop()

```
let car4 = new Car("white", "Toyota");
```

Exercise - Jewel

Create a Class with a Constructor

Your task: create a Jewel class.

The Jewel should have the following attributes:

type, price.

We want to be able to use it:

```
let goldRing = new Jewel("ring", 250);
```

Constructor basics

When writing a constructor, we can choose not to receive any parameters.

```
class Car {
    constructor(){
        this.num_of_wheels = 4;
    }
}
let myCar = new Car();
console.log(myCar.num_of_wheels); //4
```

- This constructor doesn't have any parameters
- This is good for creating default values
- It creates an object with default properties: each car will have 4 wheels.

Constructor basics

Data Formatting

The parameters that are passed to the constructor can be formatted however we want:

```
class Animal {
  constructor(name, type) {
    this.name = name.toLowerCase();
    this.type = type.toUpperCase()
  }
}

let myDog1 = new Animal("Bella", "dog");
let myDog2 = new Animal("Johnny", "dog");
```

Since this is a function we can do anything we could do in a function!

```
myDog1
Animal {name: "bella", type: "DOG"}
```

Exercise – Book Shop

Create Constructor and Instance

Your task: create a Shop class.

The shop should have the following attributes:

name, address and a default
opening hour at 8 (can be a number).

Create the following shop instance:

The shop “Little Prince” is located in King George st. 19.

Object properties

We have the Animal class:

```
class Animal(name, type, age){
   constructor(name, type){
     this.name = name;
     this.type = type;
   }
}
```

And we know how to create instances:

```
let blacky = new Animal("blacky", "dog");
let bella = new Animal("bella", "cow");
```

How do we access the instance properties?

We access the instance properties like we access object properties:

```
console.log(blacky.type);// dog
console.log(bella.type);// cow
```

Well, of course! blacky is an object!

Exercise – Expensive Jewel

Instance Properties

Your task:

Write a global function getHigherPrice
that receives 2 Jewels
and returns the higher price.

For example:

for input:

1. A 200$ ring
2. A 300$ bracelet

the function should return 300.

```
let ring = new Jewel("ring", 200);
let bracelet = new Jewel("bracelet", 300);
getHigherPrice(ring, bracelet); //300
```

Modifying an instance of an object

Instance Properties

Properties may be added to an instance of an object, without changing the constructor.

```
let myDog1 = new Animal("Bella", "dog");
myDog1.cuddle = true;

let myDog2 = new Animal("Johnny", "dog");
```

```
myDog1
Animal {name: "bella", type: "DOG", cuddle: true}
myDog2
Animal {name: "johnny", type: "DOG"}
```

The ‘this’ keyword

this is used to refer to the instance itself

- **Inside a constructor** – it refers to the instance being created.

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }

}

let bella = new Animal("bella", "dog");
bella.name;
```

Looping through properties

We can loop through properties of an object

This is a twist on a “for loop”

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }
}
let blacky = new Animal("blacky", "dog");
blacky.age = 4;
let bella = new Animal("bella", "dog");
```

key

value

```
for(let prop in blacky){
   console.log("property: " + prop + ", value: " + blacky[prop]);
}
```

```
property: name, value: blacky
property: type, value: dog
property: age, value: 4
```

Looping through properties

What happens if we loop through bella?

```
class Animal {
  constructor(name, type) {
    this.name = name;
    this.type = type;
  }
}

let blacky = new Animal("blacky", "dog");
blacky.age = 4;
let bella = new Animal("bella", "dog");
```

```
for(let prop in bella){
   console.log("property: " + prop + ", value: " + bella[prop]);
}
```

```
property: name, value: bella
property: type, value: dog
```

age property doesn't exist so it doesn't show up


Questions?


Connecting the dots

Everything's an object!

In JS – Everything is an object!

- A function is an object
- An array is an object
- An HTML DOM node is an object
- ➢ The console (console.log()) is an object

Creating objects

Creating an empty object in js can be done in 2 ways:

```
We already know:

let obj2 = {};


But we can also use the object constructor:

let obj1 = new Object();
```

Built-in constructors

- We can use a constructor to define a specific data object in JS:

```
var dateVar = new Date();
undefined
dateVar
Sun Dec 27 2015 14:01:13 GMT+0200 (Jerusalem Standard Time)
```


Questions?

OOP Cheat Sheet

Define a Class

```
class Animal {
  constructor(name, type){
    this.name = name;
    this.type = type;
  }

}
```

Create an instance

```
let blacky = new Animal("blacky", "dog");
```

Get property

```
blacky.type;
```

Update property

```
blacky.type = "cow";
```

Iterating through properties

```
for(let prop in bella){
  console.log("property: " + prop + ", value: " + bella[prop]);
}
```