QUESTION:

1. Why the use of “constructor “under a class in JavaScript, is it an instance or object.
2. As amateur in js, is it possible to learn this course even though am starting from basic

Rewire your brain for mobile first UX:

Responsive retrofitting: a quicker way to make your program mobile friendly

Mobile first: it addresses those constraints on the mobile devices not only on the size but on the bandwidth as well as ensures to concentrate on content and functionality.

How to improve UX by understanding user:

1. RELEVANT: be useful and quick
2. Reachability matters
3. Speed matters
4. Reduce image usage throughout site

What can we do to make the user design better? Make sure commonly used items are located at the bottom and keep them short and simple.

Designing a better mobile advertising user experience:

1. Keep your ads simple and sleek
2. Stop using pop-ups/pop-over
3. Make sure its unobtrusive
4. Content firs

Object-oriented programming (OOP) is a style of programming that involves separating the code into objects that have properties and method.

Three main concepts

Encapsulation: this involves keeping all the programming logic inside an object and making methods available to implement the functionality, without the outside world needing to know how it’s done

Polymorphism: this means various objects can share the same method, but also have the ability to override shared methods with a more specific implementation

Inheritance; This means we can take an object that already exists and inherit all its properties and methods. We can then improve on its functionality by adding new properties and methods.

Classes: is a blueprint for an object. Objects are created as an instance of class, and inherit all the properties and methods of the class.

Constructor Functions: Is an alternative way to create objects the function defines the properties and methods of an object

const Dice = function(sides=6){

this.sides = sides;

this.roll = function() {

return Math.floor(this.sides \* Math.random() + 1)

}}

**Note**: The parentheses are not required when instantiating a new object using a constructor function ie const redDice = new Dice;

The parentheses are required if any default arguments need to be provided eg

const whiteDice = new Dice(4);

**Note**: Each new object that’s created using this function will inherit the properties and methods defined in the function. This means that redDice will have a sides property and roll() method

Built-In Constructor Functions

Object,

 Array,

 Function

The easiest way to create a new object is to use the literal syntax: const literalObject = {};

the easiest way to create an array is to use the literal syntax, :const literalArray = [1,2,3];<< [1, 2, 3]

ES6 Class Declarations

class Dice { constructor(sides=6) {

this.sides = sides;

}

roll() { return Math.floor(this.sides \* Math.random() + 1) }}

*To create an instance of the Dice class, the new operator is again used*:

const blueDice = new Dice(20);

<< Dice { sides: 20 }

The variable *blueDice* now contains an instance of the Dice class and behaves in exactly the same way as the redDice object:

Static Methods: The static keyword can be used in class declarations to create a static method. These are sometimes called class methods in other programming languages. A static method is called by the class directly rather than by instances of the class.Static methods are not available to instances of the class. So, in our example, the instances of Dice such as redDice and blueDice cannot call the static description() method

Prototypal Inheritance: This means that every class has a prototype property that is shared by every instance of the class. any properties or methods of a class’s prototype can be accessed by every object instantiated by that class.

class Turtle {

constructor(name) {

this.name = name;

this.weapon = 'hands';

}

sayHi() {

return `Hi dude, my name is ${this.name}`;

}

attack() {

return `Feel the power of my ${this.weapon}!`;

}

}

This can then be used to create a new turtle instance:

const leo = new Turtle('Leonardo');

<< Turtle { name: 'Leonardo' }

leo.name;

<< 'Leonardo'

leo.sayHi();

<< 'Hi dude, my name is Leonardo'

The variable leo points to an instance of the Turtle class. It has a name property and a sayHi() method that references the name property.

*All objects ultimately inherit from the prototype of the Object () constructor function.*

Inheritance Using extends

A class can inherit from another class using the extends keyword in a class declaration.

class Turtle {

constructor(name) {

this.name = name;

}

sayHi() {

return `Hi dude, my name is ${this.name}`;

}

swim() {

return `${this.name} paddles in the water`;

}

}

In Previously example we adding more specific properties such as weapons that don't really apply to normal turtles, they are for ninja turtles. Instead of polluting the Turtle class with these properties, it would be a good idea to create a sub-class or child class of the Turtle class called ninjaTurtle. Notice the use of the ‘extends’ keyword:

class NinjaTurtle extends Turtle {

constructor(name) {

super(name);

this.weapon = 'hands';

}

attack() { return `Feel the power of my ${this.weapon}!` }

}

Public and Private Methods: An object’s methods are public in JavaScript. Methods and properties are said to be public because they can be queried directly and changed by assignment. We can use the concept of variable scope to keep some properties and methods private inside of a class declaration. This will prevent them from being accessed or changed. Instead, we will provide a getter method to return the values of any private properties.

Polymorphism: it means that different objects can have the same method, but implement it in different ways. The Object.prototype object has a toString() method that is shared by all objects. This means every object created in JavaScript will have a toString() method. Although every object has a toString() method, the way it’s implemented can vary between different objects.

One example of a function that uses the toString() method is the console.log() method

When an object is given as an argument to a method that isn’t a string, it will call toString() on that object in the background and display the return value in the console.

console.log([1,2,3]);

<< [ 1, 2, 3 ]

Getters and Setters

An object property descriptor can have get() and set() methods instead of a value attribute. All objects must have one or the other, they can't have both

Eg set function will prohibits a non-positive number of sides while get function that will return a description of the number of side

class Dice {

constructor(sides=6){

Object.defineProperty(this, 'sides', {

get() {

return `This dice has ${sides} sides`;

},

set(value) {

if(value > 0) {

sides = value;

return sides;

} else {

throw new Error('The number of sides must be positive');

}

}

});

this.roll = function() {

return Math.floor(sides \* Math.random() + 1)

}

}

}

Create new dice

const yellowDice = new Dice;

yellowDice.sides

<< "This dice has 6 sides"

yellowDice.sides = 10;

<< 10

yellowDice.sides

<< "This dice has 10 sides"

yellowDice.sides = 0;

<< Error: "The number of sides must be positive"

# Object methods, "this"

In js keyword ‘this’ can be used in any function, even if it’s not a method of an object

Libraries

A JavaScript library is a piece of code that provides several methods that make it easier to achieve common tasks.

Underscore & Lodash

Underscore and lodash are functions that provide additional functionality to the language

A library should not be used because of a lack of understanding of JavaScript. Instead, it should be used to speed up.