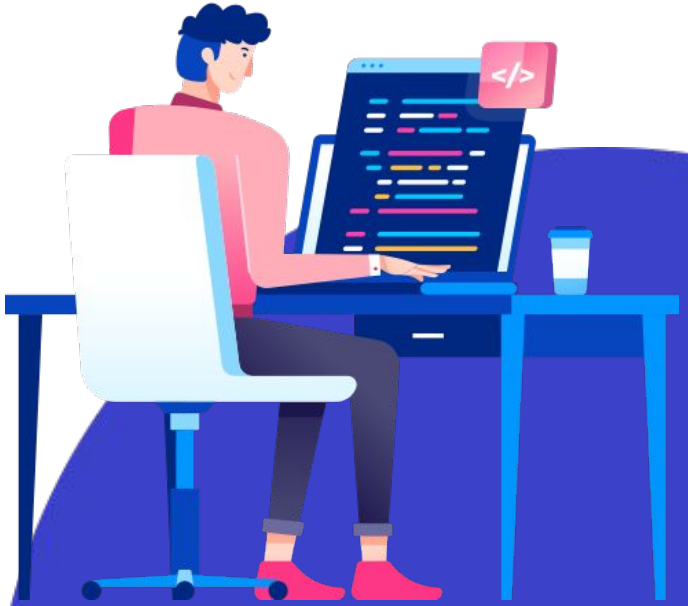
PROGRAMMING FUNDAMENTALS

Objects in JS

DIPLOMA IN FULL-STACK DEVELOPMENT

Certificate in **Computing Fundamentals**



OOP

Object Oriented
Programming

DEFINITION

Objects group together a set of variables and functions to create a model

Used to model the real world.

Image source:

LITERAL OBJECTS

```
let hotel  = {

  name: 'Raffles Hotel',
  rooms:  100,
  booked: 24,
  gym: true,
  roomTypes: ['twin','suite','delux'],

  checkAvailability: function() {
    return this.rooms - this.booked;
  }

};
```

KEY / NAME

VALUE

PROPERTIES
These are variables

METHOD
This is a function

LITERAL OBJECTS

- OBJECT
- KEY / NAME
- VALUE

```
let hotel  = {

  name: 'Raffles Hotel',
  rooms:  100,
  booked: 24,
  gym: true,
  roomTypes:
['twin','suite','delux'],

  checkAvailability: function() {
    return this.rooms - this.booked;
  }
};
```

PROPERTIES
These are variables

METHOD
This is a function

Object is the curly braces {… } and its contents
Object stored in a variable **hotel**

Separate each key from its value using a colon
Separate each property and method with a comma

The this keyword in checkAvailability() method,
References the rooms and **booked** projects of the Object (hotel)

https://replit.com/@immalcolm/js-objects#index.html

ACCESSING AN OBJECT

OBJECT

PROPERTY/METHOD NAME

```
let hotelName = hotel.name;
let roomsFree = hotel.checkAvailability();
```

MEMBER OPERATOR

```
let hotelName = hotel['name'];
let roomsFree = hotel['checkAvailability']();
```

https://repl.it/@malcolmyam/wk06-objects#script.js

UPDATING AN OBJECT

OBJECT PROPERTY NAME PROPERTY VALUE

```
hotel.name = 'Favcho Royale Hotel';
```

MEMBER OPERATOR ASSIGNMENT OPERATOR

***Note:** If the object does not have the property you are trying to update, it will be added to the object

```
hotel['name'] = 'Favcho Royale Hotel';
```

```
delete hotel.name; // Delete a property using the delete keyword
hotel.name = '';   // Clear the value of a property by assigning a blank string
```

CONSTRUCTOR NOTATION

- OBJECT
- KEY / NAME
- VALUE

```
let hotel = new Object();

hotel.name = 'Raffles Hotel';
hotel.rooms = 100;
hotel.booked = 24;
hotel.gym = true;
hotel.roomTypes =
['twin','suite','delux'];

hotel.checkAvailability: function() {
  return this.rooms - this.booked;
}
```

PROPERTIES

METHOD

Create an object using the "new" keyword and the **Object()** constructor (Blank object)

Add properties, methods to the newly created blank object

FUNCTION BASED OBJECTS

OBJECT
KEY / NAME
VALUE

PARAMETERS

```
function Hotel(name, rooms, booked){

  this.name = name;
  this.rooms = rooms;
  this.booked = 24;

  this.checkAvailability = function()
{
    return this.rooms - this.booked;
  };
}
```

PROPERTIES

METHOD

Creating a function Hotel allows it to be used as a template for creating multiple objects

This is called a function based object.

The **this** keyword is used instead of the object name to indicate the property/method belongs to the object that **this** function creates.

Each statement creates a new property or method. Uses **semi-colon** instead of comma (literal object syntax)

https://replit.com/@immalcolm/js-objects#script.js

MULTIPLE OBJECT INSTANCES

OBJECT

CONSTRUCTOR FUNCTION

```
let favchoHotel = new Hotel('Favcho Hotel', 100, 25);
let lizzieHotel = new Hotel('Lizzie Inn', 48, 12);
```

ASSIGNMENT OPERATOR

NEW KEYWORD

VALUES USED IN PROPERTIES OF THIS OBJECT

The first object **favchoHotel**. Name is "Favcho Hotel".
100 rooms, 25 booked

The second object **lizzieHotel**. Name is "Lizzie Inn".
48 rooms, 12 booked

**** Note:*** Even when multiple objects are created using the same constructor function, the methods stay the same.

https://replit.com/@immalcolm/js-objects#script.js

	KEY	VALUE
PROPERTIES:	name	string
	rooms	number
	booked	number
	gym	boolean
	roomTypes	array
METHODS:	checkAvailability	function

Objects consist of a set of key/value pairs (key can be referenced as name)

LITERAL OBJECTS

- Has no constructors
- Cannot be inherited
- Best used for “one off” objects where only one copy will exist
- A colon separates the key/value pair
- For global or configuration objects like 'game settings'

FUNCTION-BASED OBJECTS

- Has a constructor
- Can be inherited (but very messy, out of scope here)
- Best use for stuff that you want to instantiate again and again)
- The this keyword is used instead of the object name
- If you have lots of objects with similar functionality

ACTIVITY

“Objects” (30min)

OVERBAE

In a fictional Karaoke Party game called "OverBae", there are singers and judges. One singer character **"momobae"** has the following properties:

Name: Momobae
Specialty: K-Pop
Power: 49
Hitpoints: 28
Level: 7
Gender: Female

a.) Create a literal object variable named **"momobae"** that contains the above properties. For each property, indicate the appropriate data types to use.

b.) Create a function-based object method named **"Singer"** that is able to fulfil the above properties. Indicate the constructor, parameters, variables used in your function. Create a method for your function object that calculates their maximum power. This is formulated by power multiplied by level minus the current hit points.

Create a method **"singerProfile"** to allow printing of the singer statistics. Use template literals to help in your code. It should read as follows:
“Momobae Level 7, gender Female, specialty ‘K-Pop’.
Power 49!
Hitpoints: Weak.”

Create two new singer object variable **"momobae"** and “**minabae**” using the function based object you created. You may assume your values for your properties.
The hitpoints reference chart are to be followed:
<=50 = Weak
51-70 = Strong
71-100 = Amazing

ACTIVITY

“Objects” (20min)

c. Create an array to store your two new singers. Write a loop function to print out the "singerProfile" of each singer.

d. **(Challenge)**
Suggest additional **useful** properties for your objects. Or think about how you can use judges in this scenario.

VAR, LET, CONST

Understanding
Scoping

VARIABLE DECLARATION & HOISTING

```
1  function getValue(hasValue) {
2
3      if (hasValue) {
4          var value = "blue";
5          // other code
6          return value;
7      } else {
8          console.log(value);
9          // value exists here with a value of undefined
10         return null;
11     }
12     console.log(value);
13     // value exists here with a value of undefined
14 }
15 console.log(getValue(false));
16 console.log(getValue(true));
```

//hoisting happening

```
//How JS engine intepreting var
function getValue(condition) {

    var value;

    if (condition) {
        value = "blue";
        // other code
        return value;
    } else {

        return null;
    }
}
```

*NOTE:
Variable declarations using **var** are treated as if they are at the top of the function (or global scope, if declared outside of a function) regardless of where the actual declaration occurs; this is called ***hoisting***.
The scope of a variable defined with var is function scope or declared outside any function, global.

Further Reading Reference:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

LET

```
function getValue3(hasValue) {

    if (hasValue) {
        let value = "blue";
        // other code
        return value;
    } else {
        console.log(value);
        // value doesn't exist here
        return null;
    }
    console.log(value);
    // value doesn't exist here
}
console.log(getValue3(true));
console.log(getValue3(false));
```

value only exists here

```
Uncaught ReferenceError: value is not defined
```

- let declarations are not hoisted to the top of the enclosing block
- let is **block-scoped** meaning it only exist within the braces it is contained in {…}

Further Reading

https://www.w3schools.com/js/js_let.asp

VAR vs LET: Declarations

```
var course = 'IMGD';
var course = 'A3DA';
```

var allows us to re-declare a variable

```
let course = 'IMGD';
let course = 'A3DA';
```

```
Uncaught SyntaxError: Identifier 'course' has already been declared
```

let does not allow us to re-declare a variable.

LET: NO REDECLARATION

```
var count = 8;

// Syntax error
let count = 77;
```

If an identifier has already been defined in a scope, then using the identifier in a let declaration inside that scope causes an error to be thrown.

In this case, count has already been declared using **var.** The **let** declaration will throw up a syntax error. Because **let** will not redefine another variable that is in the same scope.

```
var count = 8;
// Does not throw an error
if (condition) {
    let count = 77;
    // more code
}
```

The let variable in this case creates a new variable with the same name as a variable within its scope if…..{…}

CONST

```
//valid constant
const grade = 'AD';
```

```
//syntax error: missing initialization
const grade;
```

```
Uncaught SyntaxError: Missing initializer in const declaration
```

Variables declared using **const** are considered *constants*, meaning their values cannot be changed once set.

For this reason, every **const** variable must be initialized on declaration

Declaring Objects using **const**

```
const student = {
    name : 'Joshua',
    age : 18
}
//works
student.name = 'Sean';

//works.. creating new property
student.mate = 'Nigel';

//fails
student = {
  name : 'Scott',
  age : 21
}
```

The binding student is created with an initial value of an object with two properties.

It’s possible to change student.name without causing an error because this changes what student contains and doesn’t change the value that person is bound to. It is also possible to create additional properties for the object.

Code fails when we assign student to a new value (changing the information that it is bound to).

A const declaration prevents modification of the binding and not of the value itself.

For...of

ES6 Loop

*Example: Display the word "Expense: x" using **for ..of loop***

```
let dailyExpenses = [29, 81, 43];
```

KEYWORD

VARIABLE

ITERABLE

```
for (let expense of dailyExpenses){
    expense *= 2;
    console.log("Expense:" + expense);
}
```

CLOSING CURLY BRACE
(END OF LOOP)

CODE TO EXECUTE DURING LOOP
(INSIDE THE LOOP)

Output
Expense: 58
Expense: 162
Expense: 86

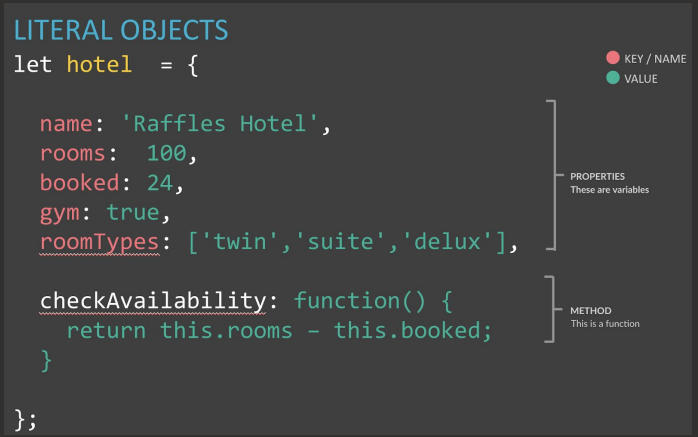
The **for...of statement** creates a loop iterating over iterable objects
- Array, Map, Set, String, TypedArray, arguments

*Reference Article
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for...of

DATA FORMATS
JSON

JSON looks like object literal syntax but it is just **data**, not an object:

```
{
  "name": "Pikachu",
  "hp": 200,
  "attack": 999,
  "defense": 999,
  "type": "electric"
}
```



https://www.w3schools.com/js/js_json_intro.asp
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

JSON data is made up of **keys** and **values**:

```
{
   "name": "Pikachu",
   "hp": "200",
   "attack": "999",
   "defense": "999",
   "type": "electric"
}
```

KEY in double quotes
Unlike your typical literal objects

VALUE

In JSON, the **key** should be placed in **double quotes** (not single quotes).

The key (or name) is separated from its value by a **colon**.

Each key/ value pair is separated by a comma. However, note that there is *no* comma after the last key/value pair

The value can be a string, number, Boolean, array, **object** or null.

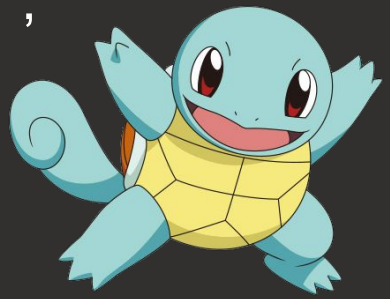
You can nest objects.

NESTING OBJECTS with JSON

```
let pokemon = [
  {
    "name":"Pikachu",
    "hp":100,"attack":9
     "defense":8,
     "color":"yellow",
     "type":"electric"
  },
  {
     "name":"squirtle",
     "hp":"47",
     "attack":"39",
     "defense":"38",
     "color":"blue",
     "type":"water"
  }
]
```

Object

Object



```
pokemon[0].name //pikachu
pokemon[1].name //squirtle
```

Image Source:
http://pokemon.wikia.com/wiki/

Summary

What have we covered?

Literal objects & Function Based Objects

For … of loop

Key Takeaway?

Practise & Practise

Use the Dev Tools/Debugger provided by the browser