Diploma Web Application Development: Introduction

ICT50220 Diploma of Information Technology(Front-End Web Development)

Code	Title
ICTWEB517	Create web-based programs
ICTWEB546	Validate application design against specifications

Sessions

- A session is a component of study
- Sessions may include:
  - Notes
  - Demonstrations
  - Challenges
  - Out of class activities

Objects

Objects

- JavaScript does not have quite the same notion of class as C#
- JavaScript Objects are similar to Python Dictionaries

Creating an Object

Each of these do the same thing:

- `const car = {}`
- `const car = new Object()`
- `const car = Object.create({})`

Objects: Properties

- Define a property in two parts:

```
propertyName : propertyValue
```

- Separate properties with a comma:

Example:

```
const game = {
  title : “Gran Turismo”,
  version:  “6”
}
```

Objects: Accessing Properties

- Access properties using the dot syntax:

  `game.title`

- Accessing an unknown property:

  returns `undefined`

- You can also use “array” type notation:

  `game['players']`

Accessing Properties

Run the following file & use case in your browser:

js/complex_object.js

```
const userObject = {
        firstName: "John",
        lastName: "Doe",
        age: 30,
        isStudent: false,
        address: {
                street: "123 Main St",
                city: "Anytown",
                zipCode: "12345"
        },
        skills: ["JavaScript", "HTML", "CSS"],
                socialProfiles: {
                twitter: "@johndoe",
                linkedIn: "linkedin.com/in/johndoe"
        },
        isEmployed: true
};
```

Setting Properties

- Setting property values when:
  - Creating the object
  - Using dot or array notation at other times

```
game.title = “Bearicades”
```

Objects: Creating New Properties

- Create new property values by using:
  - Dot, or
  - Array notation

```
game.company = “9th Level Games”
game[‘url’] = “https://www.9thlevel.com”
```

Deleting Properties

- Need to remove property?
- Use the `delete` operator

```
delete game.url
delete game['url']
```

Passing Objects

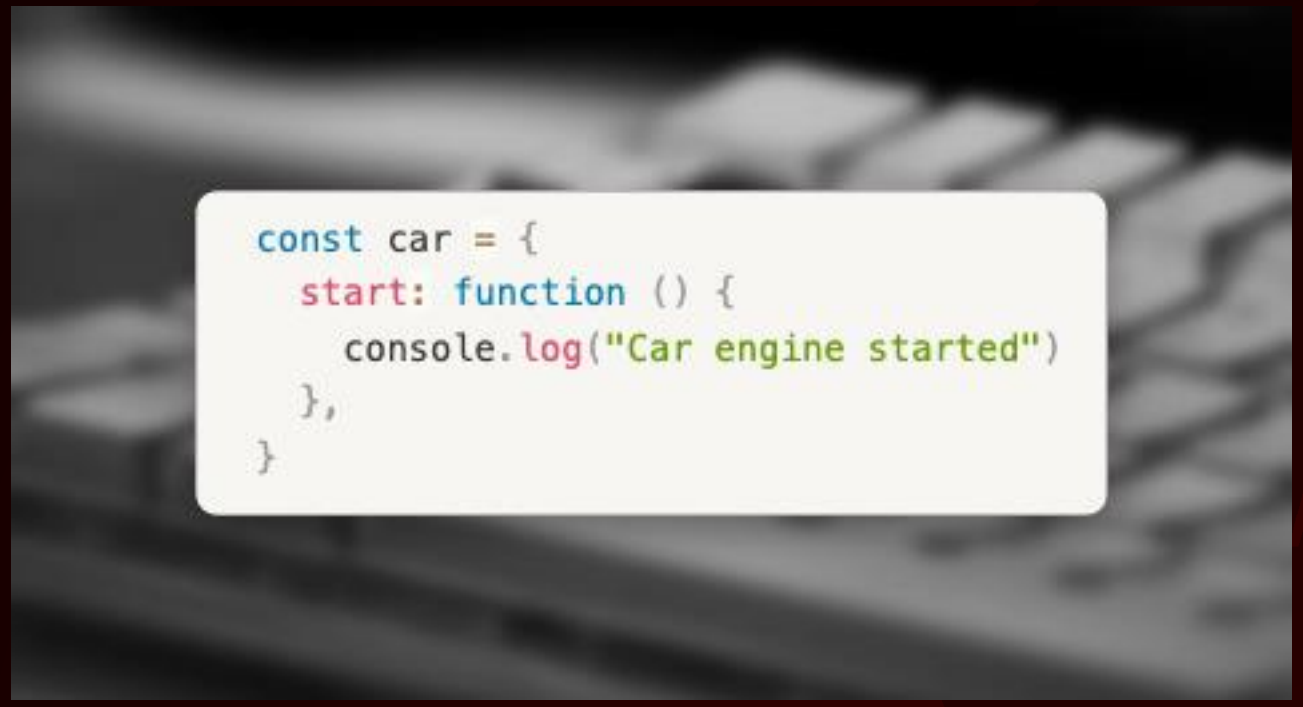
- Objects are ALWAYS passed by REFERENCE

```
let oldAge = 21
let newAge = oldAge
newAge = 32
```

- `oldAge` will still have 21

Methods

- We know about functions
- Functions may be part of an object
- They are then called methods



Objects: Methods

- Just like regular functions:
  - Methods may accept parameters
  - And may return values

Objects: Parameters

- We may **pass** and return objects from methods & functions

```
const printNameAndAge = ({ name, age }) => {
  console.log(name, age)
}

const person = {
  name: 'Georgia',
  age: 23
}

printNameAndAge(person)

//or

printNameAndAge({ name: 'Rhia', age: 19 })
```

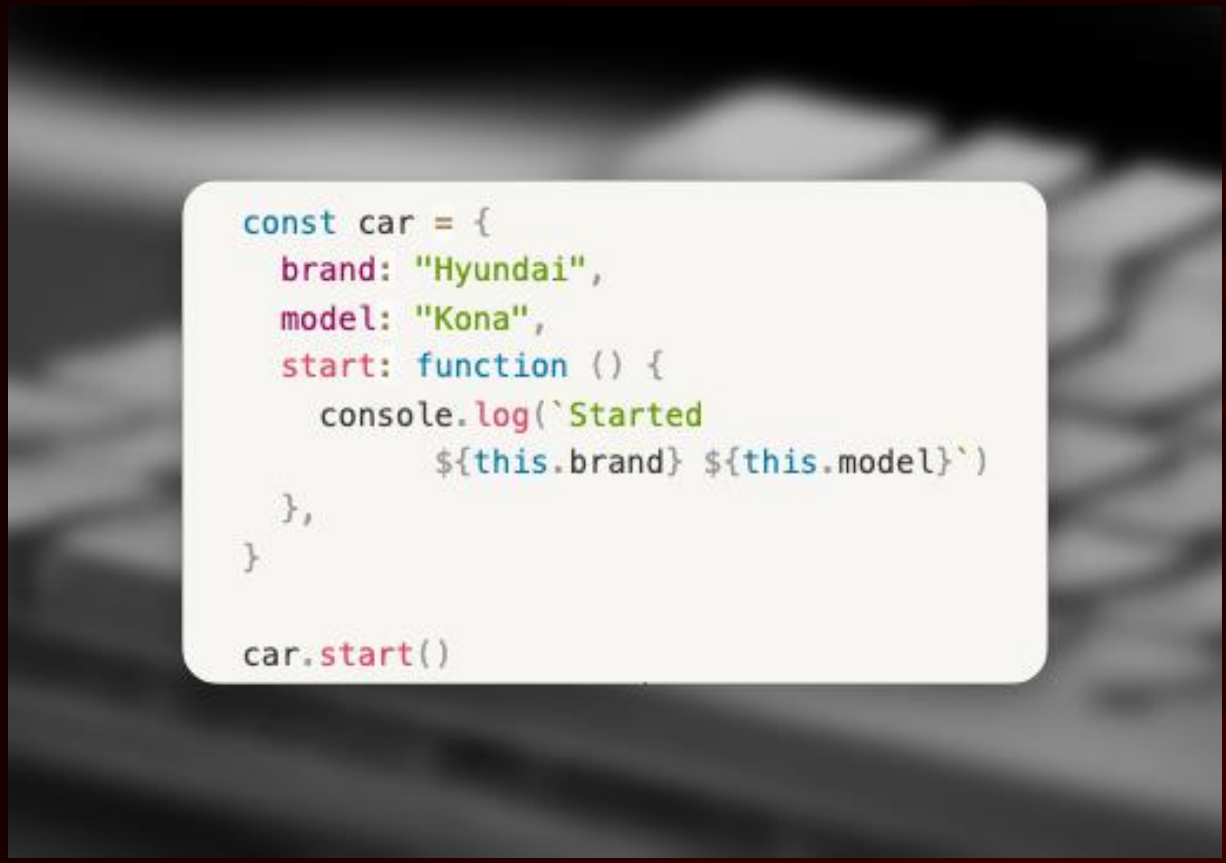
Objects: Parameters

- We may pass and **return** objects from methods & functions

- Useful when we want to return multiple values

Objects: Object Properties & Methods

- You will want to get at an object's properties when using its methods.
- The keyword `this` is employed for this purpose.



Challenge 1

Challenge 1

- Create a method **changeAge(newAge)** that takes an argument **newAge** and
- Update the **age** (hint: use **this**) property of the **userObject** with the new age.
- After defining the method, execute it with a new age value, and then log the updated **userObject** to the console.

Objects: De-structuring

Extracting some properties is possible.

```
// Extraction
const {name, age} = person
```

It is also possible to ‘rename’ the extracted property.

- Property `firstName` saved into variable `name`.

```
// Renaming
const {name: firstName, age} = person
```

Objects: Clone, Clone, Clone Again

If

```
const a = { name: 'Sue' }
```

Then

```
const b = a
```

Results in `a` and `b` pointing at the same object.

Objects: Clone, Clone, Clone Again

Deep Cloning is needed when the object has a more complex structure:

```
// Deep cloning a complex object
function deepClone(obj) {
        return JSON.parse(JSON.stringify(obj));
}
const clone1 = deepClone(userObject);
```

Objects: Clone, Clone, Clone Again

How to make a copy of an object?

Use the spread operator:

```
let newUserObject = {...userObject}
```

Objects: Sorting an Array of Objects

- It is possible to sort an array of objects
- To do so we use a property to indicate what we want to order by.

Objects: Sorting an Array of Objects

Given…

```
const list = [
  { color: 'purple', size: 'XL' },
  { color: 'red', size: 'S' },
  { color: 'black', size: '3XL' },
  { color: 'green', size: '2XL' },
]
```

Sorting on colour done using…

```
list.sort((a, b) => (a.color > b.color) ? 1 : -1)
```

Objects: Sorting an Array of Objects

Given…

```
const list = [
  { color: 'purple', size: 'XL' },
  { color: 'red', size: 'S' },
  { color: 'black', size: '3XL' },
  { color: 'green', size: '2XL' },
]
```

Sorting on colour and size done using…

```
list.sort((a, b) =>
    (a.color > b.color) ? 1 : (a.color === b.color) ? (
        (a.size > b.size) ? 1 : -1)
    : -1 )
```

Objects: Merging objects

- It is possible to merge two objects into one
- Guess which operator we use?

Objects: Merging objects

- Yes the spread!

Challenge 2

Challenge 2

- Create a function **deepClone(userObject)** that deep clones the userObject **three times** (hint: use **JSON.stringify** & **JSON.parse** to create the new object)
- Store the cloned objects in an array.
- Sort the array of cloned objects in ascending order based on the **age** property.
- Log the sorted array to the console.

Objects - Classes

Objects: JavaScript Classes

- JavaScript has a classes, but not in the typical sense of classes in other languages.
- classes are functions defined using the ***class*** keyword:

```
class ClassName {
    // properties / methods
}
```

Objects: Classes

- Example, a class for a Rectangle:

```
class Rectangle {
    const side1 = 0
    const side2 = 0
    function area() {
        return this.side1 * this.side2
    }
    function perimeter() {
        return 2 * (this.side1 + this.side2)
    }
}
```

Objects: Classes

- To use the class we create a “new” instance:
- For example, using the Rectangle class from the previous slide:
  - `myRectangle = new Rectangle()`
  - `myRectangle.side1 = 6`
  - `myRectangle.side2 = 3`
  - `console.log(myRectangle.perimeter())`

Objects: Constructor Pattern

- provides a way to create objects.
- useful when you need to create multiple objects of the same type
- each instance created with the constructor pattern has its own copy of instance-specific properties

Objects: Classes

To allow us to define properties when we create a class instance, use the constructor method:

```
class Rectangle {
    constructor (aSide, bSide) {
        this.side1 = aSide
        this.side2 = bSide
    }
    // rest of class
}
// create the instance of the class
const rectangleInstance = new Rectangle(…)
```

Challenge 3

Challenge 3

- define a **UserObject** class with a constructor that accepts all the properties present in your **userObject**.
- create an instance of the **UserObject** class, (hint: use the new keyword & provide the necessary property values.)
- log the **userObjectInstance** to the console

Web Storage API

Local & Session Storage

- In your browser's developer tools, you can access, edit, and even create cookies and entries in local storage or session storage.
- In Chromium-based browsers like Edge and Chrome, browser storage is located under the **Application tab**. In Firefox and Safari, it's located under the **Storage tab**.

Web Storage API - Methods

The API provides two methods for storing data directly in the browser:

- **window.localStorage** object for long-term storage
- **window.sessionStorage** object for transient data.

Web Storage API - Local

- use the **window.localStorage** to store data locally for use later.
- data is stored indefinitely and must be a string.

```
// Store data
let data = 'The data to store.';
localStorage.setItem('myDataKey', data);

// Get data
let savedData =
localStorage.getItem('myDataKey');

// Remove data
localStorage.removeItem('myDataKey');
```

Web Storage API - Session

- **window.sessionStorage** API works just like **window.localStorage**
- except the data is cleared when the browser session ends.

```
// Store data
let data = 'The data to store temporarily.';
sessionStorage.setItem('myTempDataKey', data);

// Get data
let tempData =
sessionStorage.getItem('myTempDataKey');

// Remove data
sessionStorage.removeItem('myTempDataKey');
```

Challenge 4

Challenge 4

In this challenge we are going to work together to build a form which saves data in the local & session storage