Application frameworks

Lab session 2 – JavaScript

Objective: Teach set of basic concepts in JavaScript programming language.

Prerequisites: Student should have basic JavaScript knowledge.

1. Understand the 'this' keyword in JavaScript.

    - Declare a variable named *vehicleName* in the window object (*var*) and assign value 'Toyota' to it. This a global variable declaration.
    - Declare a function named *printVehicleNameOuter* to print out the vehicle name. (Use *this.vehicleName*, since the variable you declared in the above step is a global variable assigned to window object. In global scope *this* points to *window* object. In the browser scope global object is *window* object and all the global variable and function declarations are assigned to *window* object).
        - Test this by printing *this* and *window* object to console. And verify that they both prints window object.
        - Check the *vehicleName* and the *printVehicleNameOuter* you declared are in the printed *window* object.
    - Call the function *printVehicleNameOuter* and verify it is printing 'Toyota'.
    - Declare an object named *Vehicle* (using object literal notation '{}') which have a variable called *vehicleName* and assigned the value 'Nissan' to it. Declare a function named *printVehicleNameInner* and assign *printVehicleNameOuter* to it.
    - Execute the *printVehicleNameInner* function and see the results.

```
var vehicleName = 'Toyota';

function printVehicleNameOuter() {
    console.log(this.vehicleName);
}

console.log(this);

printVehicleNameOuter();

var vehicle = {
    vehicleName: 'Nissan',
    printVehicleNameInner: printVehicleNameOuter
};

vehicle.printVehicleNameInner();
```

Notice how JavaScript understands the *this* keyword.

- Change the function *printVehicleNameInner* to return anonymous function (function without name) which prints *this.vehicleName (printVehicleNameInner* function body should return another function).
- Call *printVehicleNameInner* function and assigned the return value to variable named *execute*.
- Call the function *execute*. Understand that *execute* is assigned the function returned from *printVehicleNameInner* function.

```
var vehicle = {
   vehicleName: 'Nissan',
   printVehicleNameInner: function () {
        return function () {
            console.log(this.vehicleName);
        }
   }
};

var execute = vehicle.printVehicleNameInner();
execute();
```

Notice the change of values printed.

Try the functionality of 'bind' and 'call' methods in JavaScript.

- Call *execute* function by using *call* method. In *call* first argument is execution context where function should be executed. *Call* method executes the function in the passed execution context. Pass *vehicle* object as the first argument to *call* method.

```
execute.call(vehicle);
```

Notice the value that got printed.

- Now, when assigning *printVehicleNameInner* return value to *execute* call *bind* method. *Bind* method takes the execution context as the first argument and returns a new function with passed execution context bound.

```
var execute = vehicle.printVehicleNameInner().bind(vehicle);
execute();
```

Notice the value that got printed.

Further try to parameterize these functions and pass arguments using *call* and *bind* methods.

2. Understanding JavaScript closure.
   - Create a function named *taxCalculator* which accepts the tax percentage as an argument.

- *taxCalculator* should return another function which accepts the amount as an argument and returns calculated tax percentage (amount*taxPercentage/100).
- Call *taxCalculator* function and assigned the returned value to a variable.
- Now, call that variable (it is a function now) with different amounts and get tax value calculated.
- Notice how you have encapsulated the tax percentage and calculation from consumers. Now consumers can calculate the tax percentage without the knowledge of tax percentage and calculation.

```
function taxCalculator(tax) {
   var taxPercentage = tax;
   return function (amount) {
       return amount * tax / 100
   }
}

var calculator = taxCalculator(10);

console.log(calculator(90));
```

3. Write a function to call GitHub API (https://api.github.com/users) and get users.
   - Try to understand the functionality of Promises.
   - Print all users to console.

```
function fetchUsers() {
   fetch('https://api.github.com/users').then(function (response) {
       return response.json();
   }).then(function (json) {
       console.log(json);
   });
}

fetchUsers();
```

   - Try to return fetched users and print it in the caller.

```
function fetchUsers() {
   return fetch('https://api.github.com/users')
       .then(response => response.json());
}

fetchUsers().then(function (json) {
   console.log(json);
});
```

   Notice the asynchronous execution of the JavaScript and how it is resolved using promises.

4. Classes in JavaScript
   - Create a class named Vehicle using a function.
   - Add property named type to the class (*this.type*). Assign a value to that variable using a

constructor argument.

- Add a function to its prototype named drive (*Vehicle.prototype.print...*). Print 'Vehicle is driving' in the function body.
- Add VehicleCount (Vehicle.VehicleCount) as a static variable.
- Increase the number of VehicleCount (*Vehicle.VehicleCount++*) by one inside the constructor.
- Create an object from Vehicle class (*new Vehicle*) and check static variable value, type property value and function works.
- Create a class named Car and extend the class Vehicle (Car.prototype = Object.create(Vehicle.prototype); Car.prototype.constructor = Car).
- Add a new method called balanceWheels to Car and print 'Wheels are balanced' in the function body.
- Call balanceWheels and drive methods using a car object and verify the functionality.
- Check the static variable value and type variable value. Notice that they are not correct.
- The reason for the above behavior is that we didn't call the parent constructor from the child class. Do this by using the *call* method (in Car constructor function Vehicle.call(this, type);
- Re-validate the values.

```
function Vehicle(type) {
   Vehicle.VehicleCount++;
   this.type = type;
}
Vehicle.VehicleCount = 0;
Vehicle.prototype.drive = function () {
   console.log('Vehicle is driving');
};
var vehicle = new Vehicle('Toyota');
function Car(type) {
   Vehicle.call(this, type);
}
Car.prototype = Object.create(Vehicle.prototype);
Car.prototype.constructor = Car;
Car.prototype.balanceWheels = function () {
   console.log('Wheels are balanced');
};
var car = new Car('Nissan');
car.drive();
```

```
car.balanceWheels();

console.log(car.type, Vehicle.VehicleCount);
```

5. Declare variables using *const* and *let* keywords and observe the difference.

```
let vehicleName = 'Toyota';

vehicleName = 'Nissan';

const COUNTRY = 'Japan';
```

6. Use arrow functions.

```
function fetchUsers() {
   return fetch('https://api.github.com/users')
        .then(response => response.json());
}

fetchUsers().then(json => {
   console.log(json);
});
```

   Try to understand the difference between using curly brackets and not using them in arrow functions body.

7. Try exercise 3 with *async/await*.

```
async function fetchUsersAsync() {
   const response = await fetch('https://api.github.com/users');
   const json = await response.json();
   console.log(json);
}

fetchUsersAsync();
```

8. Try exercise 4 *class, extends, get, set* and *super* keywords.

```
class Vehicle {
   constructor(type) {
       Vehicle.VehicleCount++;
       this.type = type;
   }
   drive() {
       console.log('Vehicle is driving');
```

```
    }
}
Vehicle.VehicleCount = 0;
const vehicle = new Vehicle('Toyota');
vehicle.drive();
console.log(Vehicle.VehicleCount);
class Car extends Vehicle {
    constructor(type) {
        super(type);
    }
    balanceWheels() {
        console.log('Wheels are balanced');
    }
}
const car = new Car('Nissan');
car.drive();
car.balanceWheels();
console.log(Vehicle.VehicleCount);
```

Discuss the use of *static* keyword.