Лабораторная работа 3

Объекты и классы

(20 баллов)

Выполните самостоятельно следующие задания и оформите отчет.

Требования по отчету:

Наличие титульного листа. Размер страницы должен соответствовать формату А4 (210х297), размеры полей: левое – 30 мм, правое – 10 мм, верхнее – 15 мм, нижнее – 20 мм. Шрифт Times new Roman, размер 14 pt полуторный междустрочный интервал. Выравнивание текста – по ширине, красная строка – 1,25 см, отступ слева и справа – 0 мм.

Объекты

1. Выполните следующие действия:
    a. Создайте пустой объект user.
    b. Добавьте свойство name со значением John.
    c. Добавьте свойство surname со значением Smith.
    d. Измените значение свойства name на Pete.
    e. Удалите свойство name из объекта.
2. Создайте объект myBrowser со свойствами name (значение “Microsoft Internet Explorer”) и version (значение «9.0»). Выведите значения свойств на экран при помощи цикла for in.
3. Напишите функцию isEmpty(obj), которая возвращает true, если у объекта нет свойств, иначе false.
4. Можно ли изменить объект, объявленный с помощью const? Как вы думаете? Запустите и разберите следующий код:

```
const user = {
  name: "John"
};

// это будет работать?
user.name = "Pete";

// а это?
user = 123;
```

5. Создайте функцию multiplyNumeric(obj), которая умножает все числовые свойства объекта obj на 2.
6. Создайте объект calculator (калькулятор) с тремя методами:
    - read(a, b) (читать) принимает два значения и сохраняет их как свойства объекта.
    - sum() (суммировать) возвращает сумму сохранённых значений.
    - mul() (умножить) перемножает сохранённые значения и возвращает результат.

7. У нас есть объект ladder (лестница), который позволяет подниматься и спускаться:

```
let ladder = {
  step: 0,
  up() {
    this.step++;
  },
  down() {
    this.step--;
  },
  showStep: function() { // показывает текущую ступеньку
    console.log(this.step);
  }
};
```

Теперь, если нам нужно выполнить несколько последовательных вызовов, мы можем сделать это так:

```
ladder.up();
ladder.up();
ladder.down();
ladder.showStep(); // 1
ladder.down();
ladder.showStep(); // 0
```

Измените код методов up, down и showStep таким образом, чтобы их вызов можно было сделать по цепочке, например, так:

```
ladder.up().up().down().showStep().down().showStep(); // выводит 1 затем 0
```

8. Создайте функцию-конструктор класса Browser со свойствами name и version. При помощи конструктора создать объект myBrowser со значениями name = “Microsoft Internet Explorer” и version = «9.0». Вывести значения свойств на экран. Добавить к функции-конструктору класса Browser метод aboutBrowser, который будет выводить на экран информацию о свойствах этого объекта.
9. Создайте объект Сотрудник, который содержит сведения о сотруднике некоторой фирмы, такие как Имя, Отдел, Телефон, Зарплата (использовать функцию-конструктор и ключевое слово this) и отображает данные об этом сотруднике (создать метод объекта для отображения данных). Создать экземпляр объекта и вывести свойства на экран.
10. Создайте функцию-конструктор Calculator, которая создаёт объекты с тремя методами:
    - read(a, b) принимает два значения и сохраняет их в свойствах объекта.
    - sum() возвращает сумму этих свойств.
    - mul() возвращает произведение этих свойств.
11. Создайте функцию-конструктор Accumulator(startingValue). Объект, который она создаёт, должен уметь следующее:
    - Хранить «текущее значение» в свойстве value. Начальное значение устанавливается в аргументе конструктора startingValue.
    - Метод read(a) должен принимать число и прибавлять его к value.

    Другими словами, свойство value представляет собой сумму всех введённых пользователем значений, с учётом начального значения startingValue.

    Пример работы с объектом:

```
let accumulator = new Accumulator(1); // начальное значение 1

accumulator.read(10); // прибавляет 10 к текущему значению
accumulator.read(5); // прибавляет 5 к текущему значению

console.log(accumulator.value); // выведет 16
```

Работа с прототипами

1. В приведённом ниже коде создаются и изменяются два объекта. Какие значения показываются в процессе выполнения кода и почему?

```
let animal = {
  jumps: null
};
let rabbit = {
  __proto__: animal,
  jumps: true
};

console.log(rabbit.jumps); // ? (1)

delete rabbit.jumps;

console.log(rabbit.jumps); // ? (2)

delete animal.jumps;

console.log(rabbit.jumps); // ? (3)
```

2. Объект rabbit наследует от объекта animal. Какой объект получит свойство full при вызове rabbit.eat(): animal или rabbit? Продемонстрируйте это при помощи кода.

```
let animal = {
  eat() {
    this.full = true;
  }
};

let rabbit = {
  __proto__: animal
};

rabbit.eat();
```

3. У нас есть два хомяка: шустрый (speedy) и ленивый (lazy); оба наследуют от общего объекта hamster. Когда мы кормим одного хомяка, второй тоже наедается. Почему? Как это исправить?

```
let hamster = {
  stomach: [],

  eat(food) {
    this.stomach.push(food);
  }
};

let speedy = {
  __proto__: hamster
};

let lazy = {
  __proto__: hamster
};

// Этот хомяк нашёл еду
speedy.eat("apple");
console.log(speedy.stomach); // apple

// У этого хомяка тоже есть еда. Почему? Исправьте
console.log(lazy.stomach); // apple
```

4. В Javascript у вас есть возможность менять прототипы как собственных, так и встроенных классов (добавлять свойства и методы). Прототип — это объект, определяющий структуру. Например, вы можете изменить прототип класса String. Запустите и разберите следующий код:

```
// Добавление свойства по умолчанию к встроенному объекту
String.prototype.color = "black";
// Добавление (изменение) метода  к встроенному объекту
String.prototype.write = stringWrite;
function stringWrite(){
    console.log("Цвет текста: " + this.color);
    console.log("Текст: " + this.toString())
}
// используем измененный класс
let s = new String("Это строка");
s.color = "red";
s.write();
let s2 = new String("Вторая строка");
s2.write();
```

К классу String добавьте свойство size, которое хранит в себе размер шрифта. Измените метод write(), чтобы он также выводил информацию о шрифте.

5. В коде ниже мы создаём нового кролика new Rabbit, а потом пытаемся изменить его прототип. Сначала у нас есть такой код:

```
function Rabbit() {}
Rabbit.prototype = {
  eats: true
};

let rabbit = new Rabbit();

console.log(rabbit.eats); // true
```

Что будет выведено в консоль, если перед `console.log` добавить строчку:

- `Rabbit.prototype = {};`
- `Rabbit.prototype.eats = false;`
- `delete rabbit.eats;`
- `delete Rabbit.prototype.eats;`

Как вы можете это объяснить?

Классы (class)

1. Создайте класс Clock. Он должен хранить время (часы, минуты, секунды) и уметь выводить его в консоль.
2. В коде ниже класс Rabbit наследует Animal. К сожалению, объект класса Rabbit не создаётся. Что не так? Исправьте ошибку.

```
class Animal {

  constructor(name) {
    this.name = name;
  }

}

class Rabbit extends Animal {
  constructor(name) {
    this.name = name;
    this.created = Date.now();
  }
}

let rabbit = new Rabbit("Белый кролик"); // Error: this is not defined
console.log(rabbit.name);
```

3. У нас есть класс Clock. Сейчас он выводит время каждую секунду:

```
class Clock {
  constructor(template) {
    this.template = template;
  }

  render() {
    let date = new Date();

    let hours = date.getHours();
    if (hours < 10) hours = '0' + hours;

    let mins = date.getMinutes();
    if (mins < 10) mins = '0' + mins;

    let secs = date.getSeconds();
    if (secs < 10) secs = '0' + secs;

    let output = this.template
      .replace('h', hours)
      .replace('m', mins)
      .replace('s', secs);

    console.log(output);
  }

  stop() {
    clearInterval(this.timer);
  }

  start() {
    this.render();
    this.timer = setInterval(() => this.render(), 1000);
  }
}

let clock = new Clock("h m s");
clock.start();
```

Создайте новый класс ExtendedClock, который будет наследоваться от Clock и добавьте параметр precision – количество миллисекунд между «тиками». Установите значение в 1000 (1 секунда) по умолчанию.

Изменять код класса Clock запрещено.

4. Вы работаете оператором на складе. Время от времени вам привозят новые коробки. Каждая коробка имеет свою грузоподъемность w_i и объем v_i. Получая новую коробку, вы ставите на ней серийный номер, используя все целые неотрицательные числа последовательно, начиная с нуля.
   Иногда вас просят выдать коробку минимальной грузоподъемности, чтобы она выдержала предмет весом w — или коробку минимальной вместимости, в которую

можно насыпать песок объемом v. Вам нужно быстро определять серийный номер коробки, которая будет выдана. После выдачи коробки обратно на склад не возвращаются. Если подходящих коробок несколько, нужно выбрать ту, которая пролежала на складе меньше.

Нужно реализовать класс Stock, у которого, среди прочих, будет три метода:

- add(int w, int v); — добавить коробку на склад;
- getByW(int min_w); — вернуть номер коробки грузоподъемности, хотя бы min_w;
- getByV(int min_v); — вернуть номер коробки объема, хотя бы min_v.

Если подходящей коробки нет, соответствующая функция должна вернуть −1.

Продемонстрируйте и протестируйте работу всех методов класса.