|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа №1**

**по дисциплине «Архитектура ЭВМ»**

|  |  |
| --- | --- |
| **Студент:** Садулаева Т. Р.  **Группа:** ИУ7-54Б  **Преподаватели:** Попов А.Ю., Колотовкин М.И., Куклина Н.И.  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |  |

### **Цель работы**

### Изучение основных типов данных, структур и возможностей языка JavaScript.

### **Подзадача 1**

### **Задание 1**

Создать хранилище в оперативной памяти для хранения информации о детях.

Необходимо хранить информацию о ребенке: фамилия и возраст.

Необходимо обеспечить уникальность фамилий детей.

Реализовать функции:

* CREATE READ UPDATE DELETE для детей в хранилище
* Получение среднего возраста детей
* Получение информации о самом старшем ребенке
* Получение информации о детях, возраст которых входит в заданный отрезок
* Получение информации о детях, фамилия которых начинается с заданной буквы
* Получение информации о детях, фамилия которых длиннее заданного количества символов
* Получение информации о детях, фамилия которых начинается с гласной буквы

**Реализация**

"use strict";

let childMap = new Map();

function print(text) {

console.log(text)

}

function childDoesNotExistError() {

print("Child with this surname does NOT exist!\n")

}

function childAlreadyExistsError() {

print("Child with this surname already exists!\n")

}

function zeroChildrenError() {

print("There are no children\n")

}

function printChildInfo(surname, age) {

print("Surname: " + surname + "\nAge: " + age + "\n")

}

function createChild(surname, age) {

if (childMap.get(surname) === undefined) {

childMap.set(surname, age)

print("Child was added successfully!")

}

else {

childAlreadyExistsError()

}

}

function readChild(surname) {

let age = childMap.get(surname)

if (age !== undefined) {

printChildInfo(surname, age)

}

else {

childDoesNotExistError()

}

}

function updateChild(surname, newSurname, newAge) {

if (childMap.delete(surname)) {

childMap.set(newSurname, newAge)

print("Child info updated.\n")

}

else {

childDoesNotExistError()

}

}

function deleteChild(surname) {

if (!childMap.delete(surname)) {

childDoesNotExistError()

}

else {

print("Child was deleted successfully!")

}

}

function childrenAverageAge(childMap) {

let sum = 0.0

for (let age of childMap.values()) {

sum += age

}

if (sum > 0.0 && !childMap.size) {

print("Average age = " + sum / childMap.size + "\n")

}

else {

zeroChildrenError()

}

}

function childrenGetOldest(childMap) {

let reqAge = -1

let reqSurname = ""

for (let child of childMap) {

if (child[1] > reqAge) {

reqAge = child[1]

reqSurname = child[0]

}

}

if (reqAge != -1) {

printChildInfo(reqSurname, reqAge)

}

else {

zeroChildrenError()

}

}

function printChildrenWithProperty(childMap, propertyFunc, property) {

if (childMap.size === 0) {

zeroChildrenError()

}

else {

for (let child of childMap) {

if (propertyFunc(child, property)) {

printChildInfo(child[0], child[1])

}

}

}

}

function isChildSurnameStartsWith(child, letter) {

return child[0].charAt(0) === letter

}

function childrenSurnamesStartWith(childMap, letter) {

if (letter.length !== 1) {

print("It's not a letter")

}

else {

printChildrenWithProperty(childMap, isChildSurnameStartsWith, letter)

}

}

function isVowelEng(letter) {

let vowelLetters = ['a', 'e', 'i', 'o', 'u']

return vowelLetters.indexOf(letter.toLowerCase()) !== -1

}

function isVowelRus(letter) {

let vowelLetters = ['а', 'о', 'э', 'е', 'и', 'ы', 'у', 'ё', 'ю', 'я']

return vowelLetters.indexOf(letter.toLowerCase()) !== -1

}

function isChildSurnameStartsWithVowel(child, property=null) {

return isVowelEng(child[0].charAt(0)) || isVowelRus(child[0].charAt(0))

}

function childrenSurnamesStartWithVowel(childMap) {

printChildrenWithProperty(childMap, isChildSurnameStartsWithVowel, null)

}

function isLonger(string, number) {

return string.length > number

}

function isChildSurnameLongerThan(child, number) {

return isLonger(child[0], number)

}

function childrenSurnamesLongerThan(childMap, number) {

printChildrenWithProperty(childMap, isChildSurnameLongerThan, number)

}

function isNumberBelongsSegment(start, end, number) {

return number >= start && number <= end

}

function isChildAgeBelongsSegment(child, segment) {

return isNumberBelongsSegment(segment[0], segment[1], child[1])

}

function childrenAgesBelongSegment(childMap, segment) {

if (segment.length !== 2 || segment[0] > segment[1]) {

print("Segment is not valid")

}

else {

printChildrenWithProperty(childMap, isChildAgeBelongsSegment, segment)

}

}

**Проверка работоспособности**

function main() {

createChild("first", 10)

createChild("second", 11)

createChild("second", 12)

createChild("third", 12)

createChild("Abc", 23)

createChild("fourth", 12)

createChild("ёfifth", 14)

print("-\n")

updateChild("Abc", "aBC", 77)

print("-\n")

childrenAgesBelongSegment(childMap, [18, 17])

print("-\n")

childrenAverageAge(childMap)

print("-\n")

childrenGetOldest(childMap)

print("-\n")

childrenSurnamesLongerThan(childMap, 5)

print("-\n")

childrenSurnamesStartWith(childMap, 'f')

print("-\n")

childrenSurnamesStartWithVowel(childMap)

print("-\n")

deleteChild("ёfifth")

print("-\n")

childrenSurnamesStartWithVowel(childMap)

print("-\n")

readChild("ёfifth")

}

main()

**Результат выполнения**

Child was added successfully!

Child was added successfully!

Child with this surname already exists!

Child was added successfully!

Child was added successfully!

Child was added successfully!

Child was added successfully!

-

Child info updated.

-

Surname: first

Age: 10

Surname: second

Age: 11

Surname: third

Age: 12

Surname: fourth

Age: 12

Child with this surname does NOT exist!

Segment is not valid

-

There are no children

-

Surname: aBC

Age: 77

-

Surname: second

Age: 11

Surname: fourth

Age: 12

Surname: ёfifth

Age: 14

-

Surname: first

Age: 10

Surname: fourth

Age: 12

-

Surname: ёfifth

Age: 14

Surname: aBC

Age: 77

-

Child was deleted successfully!

-

Surname: aBC

Age: 77

-

Child with this surname does NOT exist!

### **Задание 2**

Создать хранилище в оперативной памяти для хранения информации о студентах.

Необходимо хранить информацию о студенте: название группы, номер студенческого билета, оценки по программированию.

Необходимо обеспечить уникальность номеров студенческих билетов.

Реализовать функции:

* CREATE READ UPDATE DELETE для студентов в хранилище
* Получение средней оценки заданного студента
* Получение информации о студентах в заданной группе
* Получение студента, у которого наибольшее количество оценок в заданной группе
* Получение студента, у которого нет оценок

**Реализация**

"use strict";

function print(text) {

console.log(text)

}

function studentDoesNotExistError() {

print("Student with this ID does NOT exist!\n")

}

function studentAlreadyExistsError() {

print("Student with this ID already exists!\n")

}

function zeroStudentsError() {

print("There are no students\n")

}

function printStudent(student) {

if (student) {

print("Group: " + String(student.group))

print("ID: " + String(student.id))

print("Grades: " + String(student.grades))

print("")

}

else

studentDoesNotExistError()

}

function printStudentGroupless(student) {

if (student) {

print("ID: " + String(student.id))

print("Grades: " + String(student.grades))

print("")

}

else

studentDoesNotExistError()

}

class Student {

constructor(group, id, grades) {

this.group = group;

this.id = id;

this.grades = grades;

this.gradesCount = grades.length;

}

averageGrade() {

let sum = 0

this.grades.forEach(function(grade, i, grades) {

sum += grade

})

if (this.grades.length === 0) {

return null

}

return sum / this.grades.length

}

}

class Students {

constructor(list) {

this.list = list

}

create(group, id, grades) {

let flag = false

this.list.forEach(function(student, i, list) {

if (student.id == id && !flag) {

flag = true

studentAlreadyExistsError()

}

})

if (!flag) {

let newStudent = new Student(group, id, grades)

this.list.push(newStudent)

print("Student was added successfully.\n")

}

}

read(id) {

let flag = false

let res = null

this.list.forEach(function(student, i, list) {

if (student.id == id && !flag) {

flag = true

res = student

}

})

return res

}

delete(id) {

let flag = false

for (let targetIndex = 0; targetIndex < this.list.length && !flag; targetIndex++) {

if (this.list[targetIndex].id == id) {

flag = true

this.list.splice(targetIndex, 1)

print("Student was deleted successfully.\n")

}

}

if (!flag) {

studentDoesNotExistError()

}

}

update(id, newID, newGroup, newGrades) {

let flag = false

if (id != newID) {

this.list.forEach(function(student, i, list) {

if (student.id == newID && !flag) {

flag = true

studentAlreadyExistsError()

}

})

}

for (let targetIndex = 0; targetIndex < this.list.length && !flag; targetIndex++) {

if (this.list[targetIndex].id == id) {

flag = true

this.list[targetIndex].id = newID

this.list[targetIndex].group = newGroup

this.list[targetIndex].grades = newGrades

print("Student was updated successfully.\n")

}

}

if (!flag) {

studentDoesNotExistError()

}

}

getStudentByID(id) {

for (let i = 0; i < this.list.length; i++) {

if (this.list[i].id == id) {

return this.list[i]

}

}

}

getStudentsByGroup(group) {

let arr = []

this.list.forEach(function(student, i, list) {

if (student.group == group) {

arr.push(student)

}

})

return arr

}

printStudentsByGroup(group) {

print("Group: " + String(group))

print("--------------------\n")

let arr = this.getStudentsByGroup(group)

arr.forEach(function(student, i, arr) {

printStudentGroupless(student)

print("====================\n")

})

}

getStudentInGroupWithMaxGradesCount(group) {

let maxCount = -1

let targetStudent = null

this.list.forEach(function(student, i, list) {

if (student.group == group && student.grades.length > maxCount) {

targetStudent = student

maxCount = student.grades.length

}

})

if (maxCount > 1)

return targetStudent

else

return -1

}

printStudentInGroupWithMaxGradesCount(group) {

print("--------------------\n")

let res = this.getStudentInGroupWithMaxGradesCount(group)

if (res !== -1)

printStudent(res)

else

print("There is no students in this group\n")

}

getStudentsWithotGrades() {

let arr = []

this.list.forEach(function(student, i, list) {

if (student.grades.length == 0) {

arr.push(student)

}

})

return arr

}

printStudentsWithotGrades() {

print("--------------------\n")

let arr = this.getStudentsWithotGrades()

arr.forEach(function(student, i, arr) {

printStudent(student)

print("=====================\n")

})

}

}

**Проверка работоспособности**

function main() {

let A = new Students([])

A.create("1", "1", [1, 2, 3, 4, 5, 1.5])

printStudent(A.read("1"))

A.create("2", "1", [1,2])

A.create("2", "2", [5])

A.create("1", "3", [])

A.create("3", "4", [])

A.printStudentsWithotGrades()

A.printStudentsByGroup("1")

print(A.getStudentByID("1").averageGrade())

A.printStudentInGroupWithMaxGradesCount("1")

A.printStudentInGroupWithMaxGradesCount("4")

A.update("4", "3", "10", [10])

A.update("4", "6", "10", [10])

printStudent(A.read("4"))

printStudent(A.read("6"))

A.delete("2")

printStudent(A.read("2"))

}

main()

**Результат выполнения**

Student was added successfully.

Group: 1

ID: 1

Grades: 1,2,3,4,5,1.5

Student with this ID already exists!

Student was added successfully.

Student was added successfully.

Student was added successfully.

--------------------

Group: 1

ID: 3

Grades:

=====================

Group: 3

ID: 4

Grades:

=====================

Group: 1

--------------------

ID: 1

Grades: 1,2,3,4,5,1.5

====================

ID: 3

Grades:

====================

2.75

--------------------

Group: 1

ID: 1

Grades: 1,2,3,4,5,1.5

--------------------

There is no students in this group

Student with this ID already exists!

Student was updated successfully.

Student with this ID does NOT exist!

Group: 10

ID: 6

Grades: 10

Student was deleted successfully.

Student with this ID does NOT exist!

**Задание 3**

Создать хранилище в оперативной памяти для хранения точек.

Неоходимо хранить информацию о точке: имя точки, позиция X и позиция Y.

Необходимо обеспечить уникальность имен точек.

Реализовать функции:

* CREATE READ UPDATE DELETE для точек в хранилище
* Получение двух точек, между которыми наибольшее расстояние
* Получение точек, находящихся от заданной точки на расстоянии, не превышающем заданную константу
* Получение точек, находящихся выше / ниже / правее / левее заданной оси координат
* Получение точек, входящих внутрь заданной прямоугольной зоны

**Реализация**

"use strict";

const eps = 1e-5

function print(text) {

console.log(text)

}

function dotDoesNotExistError() {

print("Dot with this name does NOT exist!\n")

}

function dotAlreadyExistsError() {

print("Dot with this name already exists!\n")

}

function zeroDotsError() {

print("There are no dots\n")

}

function notEnoughDotsError() {

print("There are not enough dots\n")

}

function notRectangleError() {

print("This is not a rectangle\n")

}

function printDot(dot) {

if (dot) {

dot.printIt()

print("")

}

}

class Dot {

constructor(name, x, y) {

this.name = name

this.x = x

this.y = y

}

printIt() {

print("Name: " + String(this.name))

print("( " + String(this.x) + ", " + String(this.y) + " )")

}

}

class dotCollection {

constructor(dots) {

this.dots = dots

}

printDots(dots) {

dots.forEach(function(dot, i, dots) {

dot.printIt()

print("")

})

}

printThem() {

this.printDots(this.dots)

}

isDotInCollection(name) {

let flag = false

this.dots.forEach(function(dot, i, dots) {

if (dot.name === name && !flag) {

flag = true

}

})

return flag

}

create(name, x, y) {

let flag = false

if (this.isDotInCollection(name)) {

dotAlreadyExistsError()

flag = true

}

if (!flag) {

let newDot = new Dot(name, x, y)

this.dots.push(newDot)

print("Dot was added successfully.\n")

}

}

read(name) {

let flag = false

let res = null

this.dots.forEach(function(dot, i, dots) {

if (dot.name === name && !flag) {

flag = true

res = dot

}

})

if (!res) {

dotDoesNotExistError()

}

return res

}

delete(name) {

let flag = false

for (let targetIndex = 0; targetIndex < this.dots.length && !flag; targetIndex++) {

if (this.dots[targetIndex].name === name) {

flag = true

this.dots.splice(targetIndex, 1)

print("Dot was deleted successfully.\n")

}

}

if (!flag) {

dotDoesNotExistError()

}

}

update(name, newName, newX, newY) {

let flag = false

if (name !== newName) {

if (this.isDotInCollection(newName)) {

dotAlreadyExistsError()

flag = true

}

}

for (let targetIndex = 0; targetIndex < this.dots.length && !flag; targetIndex++) {

if (this.dots[targetIndex].name === name) {

flag = true

this.dots[targetIndex].name = newName

this.dots[targetIndex].x = newX

this.dots[targetIndex].y = newY

print("Dot was updated successfully.\n")

}

}

if (!flag) {

dotDoesNotExistError()

}

}

distanceSqr(dot1, dot2) {

let dx = dot1.x - dot2.x

let dy = dot1.y - dot2.y

return dx \* dx + dy \* dy

}

distance(dot1, dot2) {

return Math.sqrt(this.distanceSqr(dot1, dot2))

}

maxDistanceDotPair() {

let maxDistanceSqr = -1

let resPair = [-1, -1]

for (let i = 0; i < this.dots.length - 1; i++) {

for (let j = i + 1; j < this.dots.length; j++) {

let distanceSqr = this.distanceSqr(this.dots[i], this.dots[j])

if (distanceSqr > maxDistanceSqr) {

resPair[0] = this.dots[i]

resPair[1] = this.dots[j]

maxDistanceSqr = distanceSqr

}

}

}

if (resPair[0] === -1) {

resPair = null

notEnoughDotsError()

}

let resDist = -1

if (maxDistanceSqr !== -1)

resDist = Math.sqrt(maxDistanceSqr)

return { resPair, resDist }

}

dotsNotFartherThanConst(dot, cnst) {

let arr = []

let cnstSqr = cnst \* cnst

for (let i = 0; i < this.dots.length; i++) {

let dSqr = this.distanceSqr(dot, this.dots[i])

if (dSqr <= cnstSqr) {

arr.push(this.dots[i])

}

}

return arr

}

dotsOnSideRelativeToCenter(side) {

let res = []

let sign = 1

switch(side) {

case "left":

sign = -1

case "right":

for (let i = 0; i < this.dots.length; i++) {

if (this.dots[i].x \* sign > 0) {

res.push(this.dots[i])

}

}

break

case "bottom":

sign = -1

case "top":

for (let i = 0; i < this.dots.length; i++) {

if (this.dots[i].y \* sign > 0) {

res.push(this.dots[i])

}

}

break

}

return res

}

dotsInRectangle(recCorner1, recCorner2) {

let res = []

let xMax = Math.max(recCorner1.x, recCorner2.x)

let xMin = Math.min(recCorner1.x, recCorner2.x)

let yMax = Math.max(recCorner1.y, recCorner2.y)

let yMin = Math.min(recCorner1.y, recCorner2.y)

if (xMax === xMin || yMax === yMin) {

notRectangleError()

return res

}

for (let i = 0; i < this.dots.length; i++) {

if ((xMin <= this.dots[i].x && this.dots[i].x <= xMax) &&

(yMin <= this.dots[i].y && this.dots[i].y <= yMax)) {

res.push(this.dots[i])

}

}

return res

}

}

**Проверка работоспособности**

function main() {

let dots = new dotCollection([])

dots.create("1", 1.0, -2.0)

dots.create("1", 5, 3)

dots.create("2", -3, 7)

dots.create("3", 1, 2)

dots.create("4", -10, -4)

dots.printThem()

dots.read("3").printIt()

printDot(dots.read("5"))

printDot(dots.read("3"))

dots.delete("2")

printDot(dots.read("2"))

dots.create("2", -3, 7)

print("---")

dots.update("2", "5", -4, 8)

printDot(dots.read("2"))

printDot(dots.read("5"))

print(dots.maxDistanceDotPair())

print(dots.dotsNotFartherThanConst({name: "6", x: 0, y: 0}, 5))

print("---")

print(dots.dotsOnSideRelativeToCenter("right"))

print(dots.dotsOnSideRelativeToCenter("left"))

print(dots.dotsOnSideRelativeToCenter("top"))

print(dots.dotsOnSideRelativeToCenter("bottom"))

print("---")

dots.create("10", -4, 3)

print(dots.dotsInRectangle({name: "7", x: -5, y: 4}, {name: "8", x: 1, y: -1}))

print(dots.dotsInRectangle({name: "7", x: -5, y: 4}, {name: "8", x: -5, y: -1}))

}

main()

**Результат выполнения**

Dot was added successfully.

Dot with this name already exists!

Dot was added successfully.

Dot was added successfully.

Dot was added successfully.

Name: 1

( 1, -2 )

Name: 2

( -3, 7 )

Name: 3

( 1, 2 )

Name: 4

( -10, -4 )

Name: 3

( 1, 2 )

Dot with this name does NOT exist!

Name: 3

( 1, 2 )

Dot was deleted successfully.

Dot with this name does NOT exist!

Dot was added successfully.

---

Dot was updated successfully.

Dot with this name does NOT exist!

Name: 5

( -4, 8 )

{

resPair: [ Dot { name: '4', x: -10, y: -4 }, Dot { name: '5', x: -4, y: 8 } ],

resDist: 13.416407864998739

}

[ Dot { name: '1', x: 1, y: -2 }, Dot { name: '3', x: 1, y: 2 } ]

---

[ Dot { name: '1', x: 1, y: -2 }, Dot { name: '3', x: 1, y: 2 } ]

[ Dot { name: '4', x: -10, y: -4 }, Dot { name: '5', x: -4, y: 8 } ]

[ Dot { name: '3', x: 1, y: 2 }, Dot { name: '5', x: -4, y: 8 } ]

[ Dot { name: '1', x: 1, y: -2 }, Dot { name: '4', x: -10, y: -4 } ]

---

Dot was added successfully.

[ Dot { name: '3', x: 1, y: 2 }, Dot { name: '10', x: -4, y: 3 } ]

This is not a rectangle

[]

**Подзадача 2**

### **Задание 1**

Создать класс Точка.

Добавить классу точка Точка метод инициализации полей и метод вывода полей на экран.

Создать класс Отрезок.

У класса Отрезок должны быть поля, являющиеся экземплярами класса Точка.

Добавить классу Отрезок метод инициализации полей, метод вывода информации о полях на экран, а так же метод получения длины отрезка.

**Реализация**

"use strict";

function print(text) {

console.log(text)

}

class Dot {

constructor(x, y, z) {

this.x = x

this.y = y

this.z = z

}

init(x, y, z) {

this.x = x

this.y = y

this.z = z

}

printIt() {

print("( " + String(this.x) + ", "

+ String(this.y) + ", "

+ String(this.z) + " )")

}

}

class Segment {

constructor(dot1, dot2) {

this.start = dot1

this.end = dot2

}

init(dot1, dot2) {

this.start = dot1

this.end = dot2

}

printIt() {

print("Start: ")

this.start.printIt()

print("End: ")

this.end.printIt()

}

length() {

let dx = this.end.x - this.start.x

let dy = this.end.y - this.start.y

let dz = this.end.z - this.start.z

return Math.sqrt(dx \* dx + dy \* dy + dz \* dz)

}

}

**Проверка работоспособности**

function main() {

let dot = new Dot

dot.init(1, -2, 3.5)

dot.printIt()

let seg = new Segment

let start = new Dot(1, 2, 3)

let end = dot

seg.init(start, end)

seg.printIt()

print(seg.length())

}

main()

**Результат выполнения**

( 1, -2, 3.5 )

Start:

( 1, 2, 3 )

End:

( 1, -2, 3.5 )

4.031128874149275

### **Задание 2**

Создать класс Треугольник.

Класс Треугольник должен иметь поля, хранящие длины сторон треугольника.

Реализовать следующие методы:

* Метод инициализации полей
* Метод проверки возможности существования треугольника с такими сторонами
* Метод получения периметра треугольника
* Метод получения площади треугольника
* Метод для проверки факта: является ли треугольник прямоугольным

**Реализация**

"use strict";

function print(text) {

console.log(text)

}

const eps = 1e-6

class Triangle {

constructor(len1, len2, len3) {

this.len1 = len1

this.len2 = len2

this.len3 = len3

}

init(len1, len2, len3) {

this.len1 = len1

this.len2 = len2

this.len3 = len3

}

isPossible() {

let a = this.len1

let b = this.len2

let c = this.len3

return (a + b) > c && (a + c) > b && (b + c) > a

}

perimeter() {

return this.len1 + this.len2 + this.len3

}

square() {

let a = this.len1

let b = this.len2

let c = this.len3

let p = (a + b + c) / 2

return Math.sqrt(p \* (p - a) \* (p - b) \* (p - c))

}

isRectangular() {

let a = this.len1

let b = this.len2

let c = this.len3

return Math.abs(a \* a - b \* b + c \* c) < eps ||

Math.abs(b \* b - c \* c + a \* a) < eps ||

Math.abs(c \* c - a \* a + b \* b) < eps

}

}

**Проверка работоспособности**

function main() {

let T = new Triangle

T.init(3, 4, 5.000000001)

print(T.isPossible())

print(T.perimeter())

print(T.square())

print(T.isRectangular())

}

main()

**Результат выполнения**

true

12.000000001

6.000000000000001

true

**Задание 3**

Реализовать программу, в которой происходят следующие действия:

Происходит вывод целых чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Потом опять происходит вывод чисел от 1 до 10 с задержками в 2 секунды.

После этого происходит вывод от 11 до 20 с задержками в 1 секунду.

Это должно происходить циклически.

**Реализация**

"use strict";

const sec = 1000

function print(text) {

console.log(text)

}

function f() {

for (let n = 1; n <= 10; n++) {

setTimeout(print, 2 \* (n - 1) \* sec, n)

}

for (let n = 11; n <= 20; n++) {

setTimeout(print, (n + 9) \* sec, n)

}

}

function main() {

f()

let interval = setInterval(() => f(), 30 \* sec)

}

main()

**Проверка работоспособности**

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

1

...

Числа выводятся на экран согласно заявленным промежуткам времени.

**Вывод**

Язык программирования JavaScript позволяет решать задачи хранения структур данных в оперативной памяти компьютера. Ввод информации, её преобразование и выдача пользователю могут быть осуществлены с использованием таких базовых инструментов языка JavaScript, как структуры, массивы и классы. Язык JavaScript также предоставляет эффективные встроенные методы решения математических задач.