Классы в js не существуют. Но есть конструктор (класс на минималках). Любая функция в js вызванная через оператор new возвращает созданный внутри неё контекст в виде объекта. Каждый раз при использовании new каждый объект будет новый (экземпляр). Экземпляр не будет создан и контекст не будет возвращен если функция не имеет контекста.

function f1 () {…}

var a = new f1(); (a – obj)

Объект, который возвращается конструктором доступен через ключевое слово this внутри тела этой функции (конструктора). Можно передать любые свойства.

function f1 () {

var a=1;

this.b = 7;

}

var g = new f1 ();

var f = new f1;

f.b -> 7

g.b -> 7

Таким образом в конструкторе можно реализовать понятие публичных и приватных свойств переменных.

Правила объявления:

* Если функция используется в качестве конструктора, то её название необходимо писать с заглавной буквы.
* Каждый конструктор должен лежать в отдельном файле, название которого совпадает с названием конструктора, даже если объявление конструктора занимает одну строку, в этом файле нельзя создавать экземпляры этого конструктора.

Так как конструктор — это функция, то конструктор можно проинициализировать необходимыми нам данными.

function Room (id, title, description, actions) {

this.id = id;

this.title = title;

this.description = description;

this.actions = actions;

var rooms = {

start: new Room (…)

...

};

Поскольку конструктор — это функция внутри неё можно объявлять другие типы данных(функции), также можно конструктор вписать в контекст конструктора и тогда он станет его методом.

Перепишем проект графиков.

Graph.js

Function Graph (options) {

options = options || {};

var id = options.id;

var width = options.width || 300;

var height = options.height || 300;

var WIN = options.WIN || {};

var canvas;

if (id) {

canvas=document.getElementById(id);

} else {

canvas = document.createElement(‘canvas’);

document.querySelector(‘body’).appendChild(canvas);}

canvas.width = width;

canvas.height = height;

var context = canvas.getContext(‘2d’);

// у канваса как и у любого другого имеющего размеры элемента есть система событий с которыми он умеет работать в частности движение мышки в канвасе(mouseMove) и нажатие мыши(mouseup, mousedown). Обработчики этих событий будут внешними и называется это callBacking. callBack это функция которая аргументом передана в другую функцию и может быть внутри неё вызвана.

Var callbacks = options.callbacks;

canvas.addEventListener(‘wheel’, callbacks.wheel);

var PI2 = 2 \* Math.PI;

// обычно сначала описываются приватные методы

function xs(x){…}

function ys(y){…}

this.clear = function(){

context.fillStyle=’#efe’;

context.fillRect(0,0,canvas.width,canvas.height);

}

this.line = function(x1,y1,x2,y2,color,width){

context.beginPath();

context.strokeStyle=color || ‘red’;

context.lineWidth = width || 2;

context.moveTo(xs(x1),ys(y1));

context.lineTo(xs(x2),ys(y2));

context.stroke();

context.closePath();}

this.point=function(x,y,color,size){

context.beginPath();

context.strokeStyle = color || ‘#f00’;

context.arc(xs(x),ys(y),size||2,0,PI2);

context.stroke();

context.closePath();

}

}

main.js

f(x){…}

window.onload=function(){

var WIN = {…};

var graph = new Graph({id:’canvas’,

width:600,

height:600,

WIN:WIN,

callbacks:{wheel:wheel}});

var zoomStep=0.2;

// у любого обработчика есть свойство event

function wheel(event){

var delta = (event.wheelDelta>0)

-zoomStep:zoomStep;

WIN.WIDTH += delta;

WIN.HEIGHT += delta;

WIN.LEFT -= delta/2;

WIN.BOTTOM -= delta/2;

render();

}

printOXY(…)

printFunction(f,n)…

graph.line(x,f(x),x+dx,f(x+dx),color,2);

function render(){

graph.clear();

printOXY();

printFunction(f,200);}

render();

};

Домашнее задание:

Реализовать конструктор граф, масштабирование графика, и у нарисованного графика отмечать его нули.

\*Научить перетаскивать график мышкой