Лабораторная работа № 9-10. Движение объектов и графика на JavaScript. Создание меню.

Цель: изучить возможности управления процессами во времени: методы **setInterval()** и **setTimeout()**; познакомиться с приемами реализации перемещения и движения объектов; научиться отображать графики. Изучить способ создания меню на **JavaSript** .

Теория

Управление процессами во времени

Для организации постоянного периодического (через заданный интервал времени) выполнения некоторого выражения или функции служит метод **setInterval()** объекта window. Этот метод имеет следующий синтаксис:

```
setInterval (выражение, период [, язык]).
```

Первый параметр – строка, содержащая выражение (обычно вызов функции). Второй параметр – целое число, указывающее временную задержку в миллисекундах. Третий, необязательный параметр, указывает язык, на котором написано выражение; по умолчанию – JavaScript. Метод **setInterval()** возвращает некоторое целое число – идентификатор временного интервала, который может быть использован в дальнейшем для прекращения процесса.

В следующем примере функция myfunc() выполняется периодически через 0,5 с.

```
setInterval("myfunc()", 500)
```

Для остановки запущенного временного процесса служит метод **clearInterval(идентификатор)**, который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом setInterval(), например:

```
var myproc = setInterval("myfunc () , 100")
if (confirm("Прервать процесс ?"))
clearInterval(myproc)
```

Чтобы выполнить выражение с некоторой временной задержкой, используется метод **setTimeout()** объекта window, который имеет следующий синтаксис:

```
setTimeout(выражение, задержка [, язык]).
```

Первый параметр – строка, содержащая выражение (обычно вызов функции). Второй параметр – целое число, указывающее временную задержку в миллисекундах. Третий, необязательный параметр указывает язык, на котором написано выражение; по умолчанию – JavaScript. Метод setTimeout() возвращает некоторое целое число – идентификатор временного интервала, который может быть использован в дальнейшем для отмены задержки выполнения процесса.

Для отмены задержки процесса, запущенного с помощью метода setTimeout(), используется метод **clearInterval(идентификатор)**, который принимает в качестве параметра целочисленный идентификатор, возвращаемый соответствующим методом setTimeout.

В следующем HTML-документе имеются две кнопки. Щелчок на кнопке Пуск открывает через 5 с новое окно и загружает в него документ mypage.htm. Это действие можно отменить с помощью кнопки Отмена, если щелкнуть на ней, пока окно еще не открыто:

```
<HTML>
<BUTTON ID="start">Пуск</BUTTON>
<BUTTON ID="stop">Отмена</BUTTON>
<SCRIPT>
var myproc
function start.onclick(){
myproc = setTimeout ("window.open('mypage.htm')", 5000)
}
function stop.onclick(){
clearTimeout(myproc)
}
</SCRIPT>
</HTML>
```

Перемещение графических объектов

Перемещать мышью изображения можно различными способами. Изучим один из них: пользователь пытается перетащить мышью изображение; затем он должен отпустить кнопку мыши и переместить указатель в нужное место; остановившись в нужном месте, пользователь отпускает кнопку мыши или щелкает ею, чтобы прекратить перемещение изображения. Код перемещения изображения мышью:

```
<HTML>
<HEAD><TITLE>Перемещаемое изображение</TITLE></HEAD>
<BODY id = "mybody">
<IMG ID="myimg" SRC = "pict.gif ondragstart = "drag()" style
= "position:absolute; top:10; left:10">
</BODY>
<SCRIPT>
flag = false    // нельзя перемещать
var id_img = ""
function drag() {
  flag = mag
  id_img = event.srcElement.id
}
function mybody.onmousemove(){
  if (flag){     // если можно перемещать
  document.all[id_img].style.top = event.clientY
  document.all[id_img].style.left = event.clientX } }
function mybody.onmouseup(){
  flag = false     // нельзя перемещать
}
```

```
</SCRIPT>
</HTML>
```

Здесь функция drag(), обрабатывающая событие ondragstart (попытка перетаскивания), устанавливает переменную-триггер flag и выясняет, кто инициатор события. Значение переменной flag позволяет определить, можно или нельзя перемещать элемент. В данном примере инициатором события может быть только один элемент. События onmousemove (перемещение указателя мыши) и onmouseup (кнопка мыши отпущена) получает не изображение, а объект тела документа mybody.

Перемещение текстовых областей

Рассмотрим пример HTML-документа, в котором можно перемещать текстовые области, созданные с помощью тегов <TEXTAREA>. Размеры области и шрифта текста определяются в ней в зависимости от значения вертикальной координаты.

```
<HTML>
<HEAD><TITLE>Перемещаемые текстовые облас-ти</TITLE></HEAD>
<BODY id = "mybody" background = "blue.gif">
<TEXTAREA ID="tl" ondblclick=""drag()" STYLE =
"position:absolute;
top:10; left: 10; fon-size: large"> Это - первый
текст</TEXTAREA>
<TEXTAREA ID="t2" ondblclick=""drag()" STYLE =
"position:absolute;
top:100; left:150"> Это - второй текст </TEXTAREA>
<TEXTAREA ID="t3" ondblclick=""drag()" STYLE =
"position:absolute; top:150; left:250"> Это - третий текст
</TEXTAREA>
</BODY>
<SCRIPT>
resizetext() { // установка размеров текстовых областей
var flag = false }
var id_img = ""
function drag() {
flag = !flag
id_img = event.srcElement.id //id элемента, который надо
перемещать
function mybody.onmousemove() {
if (flag){
document.all[id_img].style.top = event.clientY
document.all[id_img].style.left = event.clientX
resizetext() }} // установка размеров текстовых областей
function mybody.onmouseup() {
flag = false }
function resizetext () { // установка размеров областей
var y, size, idimg, idtext
for (i =0 ; i < document.all.length; i++) {
if (document.all[i].tagName=='TEXTAREA') {
```

```
idtext = document.all[i].id
y = parseInt(document.all[idtext].style.top)
size = Math.min(y, 800)
size = Math.max(size, 60)
document.all[idtext].style.width = size
document.all[idtext].style.height = 0.8*size
document.all[idtext].style.zIndex = y
document.all[idtext].style.fontSize = Math.max(2, y/10)
</SCRIPT>
</HTML>
```

Схема сценария, осуществляющего непрерывное перемещение видимого элемента документа, имеет следующий вид:

```
    Function init_move() {  // инициализация движения
    ...     // подготовка к запуску функции move()
    setInterval("move()", задержка)
    function move(){
    .../* изменение координат top и left стиля перемещаемого
элемента */
    // вызов функции для перемещения элемента
```

Создаются две функции: **init_move(),** которая осуществляет подготовку исходных данных и вызывает метод **setInterval()** с указанием в качестве первого параметра имени второй функции **move()** в кавычках, и **move()**, которая изменяет координаты элемента. Поскольку метод setInterval() вызывает функцию move() периодически через заданное количество миллисекунд, то координаты элемента изменяются постоянно. При этом создается эффект движения. Скорость и плавность движения зависят от величин приращения координат (в функции move()) и временной задержки (второго параметра метода setInterval()). Чтобы начать перемещение элемента надо вызвать первую функцию init_move().

Остановка движения

Сценарий с запуском и остановкой движения может выглядеть, например, следующим образом:

```
    function init_move(xid, dx, dy) {
    var prmstr = "'"+xid+" ,"+dx+" ,"+dy   // строка параметров
для move()
    prmstr = "move(" +prmstr+ ")"
    idjnove = setInterval (prmstr , 200)    /* сохраняем
идентификатор движения */
    }
    function move(xid, dx , dy) {
    y = parseInt(document.all[xid].style.top)
    x = parseInt(document.all[xid].style.left)
    document.all.myimg.style.top = y+dy
    document.all.myimg.style.left = x+dx
```

```
    if    (parseInt(document.all[xid].style.left)    >    350)    /*
остановка по условию */
    }
    clearInterval(id move)
    init_move("myimg", 10, 5)     // начинаем движение
```

В этом примере движение остановится, как только горизонтальная координата элемента превысит 350 пикселов.

Движение по произвольной кривой

Рассмотрим задачу организации движения видимого элемента по произвольной кривой, заданной выражением с одной переменной. Функция движения в качестве параметров будет иметь два выражения, которые описывают изменения вертикальной и горизонтальной координат элемента. Эти выражения будут содержать одну переменную, которую мы обозначим через x – строчной латинской буквой. Переменную x можно интерпретировать как независимый параметр движения (например, время). С помощью встроенной функции eval() можно вычислить значения этих выражений при конкретном значении переменной x и присвоить их параметрам left и top таблицы стилей перемещаемого элемента. Функция (пусть это будет move()), которая все это выполняет, передается в качестве первого параметра методу setInterval(), который периодически вызывает ее через заданный интервал времени.

Функция инициализации движения **curvemove()** принимает три строковых параметра (ID перемещаемого элемента, выражение для вертикальной координаты и выражение для горизонтальной координаты) и один числовой параметр (период времени, через который координаты элемента пересчитываются). Ниже приводятся определения функций curvemove() и move():

```
function curvemove(xid, yexpr, xexpr, ztime) {
/* Движение по произвольной кривой.*/

if (!xid) return null
if (!yexpr) yexpr = "x"
if (!xexpr) xexpr = "x"
if (!ztime) ztime = 100  // интервал времени, мс
x = 0     /* глобальная переменная, входящая в выражения yexpr
и xexpr */
setInterval("move('"+xid+" ', + yexpr + " "+ xexpr +"' ztime)
}
function move(xid, yexpr, xexpr) {
  x++
  document.all[xid].style.top = eval(yexpr)
  document.all[xid].style.left = eval(xexpr)
}
```

Параметры:

– xid – id движущегося объекта, строка;

– yexpr – выражение для вертикальной координаты;
– xexpr – выражение для горизонтальной координаты;
– ztime – интервал времени между вызовами функции move(), мс.

Чтобы сделать переменную x глобальной, не надо использовать ключевое слово var перед первым ее появлением в коде.

Ниже приведен пример HTML документа с движущимся изображением:

```
<HTML>
<IMG ID = "myimg" SRC = "pict1.gif" STYLE = "position:absolute">
<SCRIPT>
function curvemove(xid, yexpr, xexpr, ztime) {
// код определения функции
}
function move(xid, yexpr, xexpr) {
// код определения функции
}
Curvemove()'myimg' , "100 + 50*Math.sin(0.03*x)" , "50 + x", 100)
</SCRIPT>
</HTML>
```

В этом примере изображение будет перемещаться по синусоиде с 50 пикселов и горизонтальной скоростью 10 пикселов в секунду. Начальные ординаты графического объекта равны 100 и 50 пикселов по вертикали и горизонтали соответственно.

Рисование линий

В JavaScript нет специальных встроенных средств для рисования произвольных линий. Для решения этой задачи ужно вывести на экран изображение размером 1x1 пиксел, залитое цветом, отличающимся от цвета фона. Это изображение следует разместить несколько раз в соответствии с координатами, которые задаются параметрами позиционирования top и left атрибута STYLE тега <IMG>. С помощью сценария можно сформировать строку, содержащую теги <IMG> с необходимыми атрибутами, а затем записать ее в документ методом write().

Рисование прямой линии

Для отображения точки в HTML можно использовать следующий тег:

```
<IMG SRC = "point.bmp" STYLE = "position:absolute; top:y; left:x">
```

Здесь point.bmp – имя графического файла, содержащего один пиксел; y, x – числа, указывающие положение графического файла в пикселах. Изображение точки размером 1x1 пиксел можно создать в любом графическом редакторе. Из соображений экономии его лучше всего

сохранить в файле формата BMP, а не JPEG или GIF (при малых размерах изображения алгоритмы сжатия неэффективны).

Чтобы задать размеры отображения точки на экране, следует использовать атрибуты WIDTH и HEIGHT (ширина и высота):

```
    <IMG SRC = "point.bmp" STYLE = "position: absolute; top:y;
left:x" WIDTH=n HEIGHT=n>
```

Одинаковые значения атрибутов WIDTH и HEIGHT задают представление точки в виде квадрата размером n×n пикселов. При этом точка с исходными размерами 1x1 пиксел просто растягивается. Таким образом, имеется возможность задать отображаемые размеры (масштаб) одной точки, а, следовательно, и определить толщину линии.

Далее следует пример определения функции, которая рисует прямую линию с заданными координатами xl, yl, x2, y2 и толщиной линии n.

```
    Function line(xl, yl, x2, y2 , n){
    /*xl, yl – начало линии, x2, y2 – конец, n – толщина */
    var clinewidth = " WIDTH=" + n + "HEIGHT=" + n  /* строка для
учета толщины */
    var xstr = "" // строка тегов для записи в HTML-документ
    var xstr0 = ' <IMG SRC="point.bmp"' + clinewidth + ' STYLE =
"positlon:absolute; '
    var k = (y2 – yl)/(x2 – xl)    // коэффициент наклона линии
    var x = xl      // начальное значение координаты x
    /* Формирование строки, содержащей теги <IMG. . . >: */
    while (x <= x2) {
    xstr += xstr9 + 'top:' + (yl + k* (x – xl)) + ': left:[1] + x +
'
    x+ +
    // запись в документ
```

Рисование кривой линии

Функция **curve()** для рисования кривой приниматn следующие параметры: имя графического файла с изображением точки (может быть любое изображение), выражение, задающее кривую, координаты начала линии, количество точек линии, толщина линии и длина штриха (если потребуется штриховая линия). Далее следует код функции curve() для рисования кривых.

```
    function curve(pic, yexpr, x0, y0, t, n, s){
    /* pict_file - имя графического файла
    yexpr - выражение с переменной x
    x0, y0 - координаты начала кривой
    t - количество точек кривой (значений переменной x)
    n - толщина линии
    s - длина штриха и паузы */
    if (!pic) pic = "red.bmp"
    var clinewidth = ""
```

```
    if (!n)
      clinewidth = 'WIDTH=' + n + 'HEIGHT=' + n
    var x
    xstr0  =  '<IMG  src="'+pic+'"  width=10px  height=10px'  +
clinewidth + ' STYLE = "position:absolute; top: '
    xstr = ""
    var i = 0,
    draw = true
    for(x = -1 * t / 2; x < t/2; x++)
    {
    if (draw)
    xstr  +=  xstr0  +  (y0  +  eval(yexpr))  +  ';  left:'  +  (x0  +
x)+'">'
    if (i > s&&s > 0)
    {
    draw = false
    i = 0
    }
    i ++
    }
    document.write (xstr)
    }
```

Вызов функции:

```
curve("yellow.bmp", "200-0.01*x*x", 500, 100, 300, 3, 0)
```

Создание меню при помощи HTML/CSS/JS

При обработке события браузер автоматически передает в функцию обработчика в качестве параметра объект **Event**, который инкапсулирует всю информацию о событии. И с помощью его свойств может быть получена информация:

- **target:** указывает на элемент, на котором было вызвано событие
- **cancelable:** возвращает true, если можно отменить стандартную обработку события
- **currentTarget:** определяет элемент, к которому прикреплен обработчик события
- **defaultPrevented:** возвращает true, если был вызван у объекта Event метод preventDefault()
- **eventPhase:** определяет стадию обработки события

и др.

Примеры:

Index.html

```
<body>
     <div id="nav">
```

```
     <div class="menu-item">
     Меню 1
     <div class="submenu">
     Подменю
     </div></div>
     <div class="menu-item">
     Меню 2
     <div class="submenu">
     Подменю
     </div></div>
     <div class="menu-item">
     Меню 3
     <div class="submenu">
     Подменю
     </div></div>
     </div>
     <script src="script.js"></script>
</body>
```

Style.css

```
.menu-item{
     border: 2px soild #66CDAA;
     display: inline-block;
     position: relative;
     cursor: pointer;
}

.menu-item .submenu{
     height: 40px;
     width: 120;
     background: green;
     position: absolute;
     cursor: pointer;
}
```

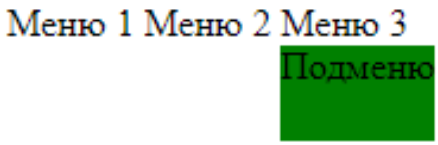
Script.js

```
document.getElementById('nav').onmouseover = function(event){
//отслеживает нахождение мыши внутри блока
     var target = event.target;
//отслеживаем клик
     if (target.className == 'menu-item'){
          var s = target.getElementsByClassName('submenu');
          closeMenu();
          s[0].style.display='block';
//массив, содержащий ложное меню
     }
}
```

```
document.onmouseover=function(event){
     var target = event.target;
     console.log(event.target);
     if (target.className!='menu-item' &&
target.className!='submenu'){
          closeMenu();
     }
}

function closeMenu(){
//получение всех элементов подменю в блоке nav, присваивание
display:none
     var menu = document.getElementById('nav');
     var subm=document.getElementsByClassName('submenu');
     for (var i=0; i <subm.length; i++){
          subm[i].style.display ="none";
     }
}
```

В результате получится меню как на изображении. При наведении курсора на один из пунктов, появляется подменю.



Задания к лабораторной работе № 9

Задание 1. Создать программу на JS, позволяющую перетащить мышью область текста и картинку.

Задание 2. Создать программу на JS, организующую движение картинки по прямой линии.

Задание 3. Создать программу на JS, организующую движение картинки по кривой линии и возврат ее в исходное состояние.

Задание 4. Нарисовать график функции, выбрав функцию с помощью радио кнопки ($y=x^2$, $y=x^3$, $y=\sin(x)$, $y=\cos(x)$).

Задание 5*. Реализовать выбор цвета графика функции с помощью списка.

Задание 6. Создайте меню, использую средства HTML/CSS/JS

Порядок выполнения:

- Изучите компоненты кода, представленные в теоретической части;

- Соберите из них код;
- Ознакомьтесь с комментариями в коде JS;
- Найдите 3 ошибки в script.js;
- Усовершенствуйте код добавлением элементов графической визуализации (Подробнее в лабораторной работе 1.6).