# WEB-технологии в дизайне

HTML 5 CANVAS

# Canvas

HTML 5 Canvas предоставляет простой и мощный способ вывода графики и рисования с использованием JavaScript. Для каждого элемента canvas можно использовать контекст, в котором нужно вызвать команды JavaScript для рисования на Canvas. Браузеры могут реализовывать несколько кон-текстов элемента canvas и предоставлять различные API для рисования. Рисование происходит в растровой форме, то есть, нарисовав на канве какую-либо фигуру, её нельзя будет изменить или удалить отдельно, — можно только стереть целую область канвы.

# Контексты

Большинство современных браузеров предоставляют возможности 2D-контекста (2D Canvas) — Opera, Firefox, Konqueror и Safari. Кроме того существуют эксперимен-тальные сборки браузера Opera, которые включают поддержку 3D-контекста (3D Canvas), а также дополнение к Firefox, которое реализует поддержку 3D Canvas.

# Основы использования Canvas

Необходимо добавить идентификатор к элементу canvas, чтобы потом обратиться к нему в JavaScript, также необходимо задать атрибуты width и height для определения ширины и высоты элемента canvas.

```
<canvas id="myCanvas" width="300" height="150">
Альтернативное содержимое, которое будет показано, если браузер не поддерживает Canvas.
</canvas>
```

# Рисование внутри элемента canvas

Сначала нужно найти созданный тег canvas с помощью функции getElementById, а потом инициализировать нужный контекст. После можно начинать рисование на канве, используя доступные API-команды выбранного контекста.

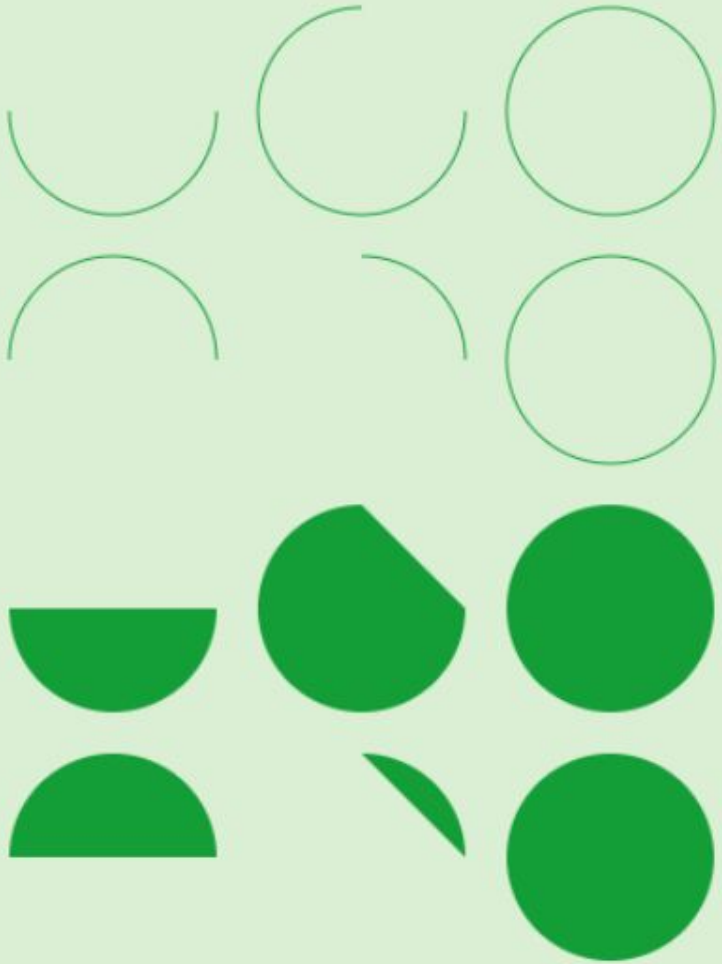
```
var elem = document.getElementById('myCanvas');
   if (elem && elem.getContext)
   {
     // Получить 2D контекст.
       var ctx = elem.getContext('2d');
            if (ctx)
           {
              // Команды API  для рисования фигур
           }
   }
```

# Canvas 2D API

- beginPath() и closePath()
- moveTo(x, y)
- lineTo(x, y)
- rect(x, y, width, height)
- fillRect (x1, y1, x2,y2)
- clearRect(x, y, width, height)
- arc(x, y, R, α , β, [направление: true - против часовой])
- ellipse (x, y, Rx, Ry, вращение, α, β, [против часовой])
- arcTo(x1, y1, x2, y2, r)
- quadraticCurveTo(x1, y1, x, y)
- bezierCurveTo(x1, y1, x2, y2, x, y)

# Пример использования arc

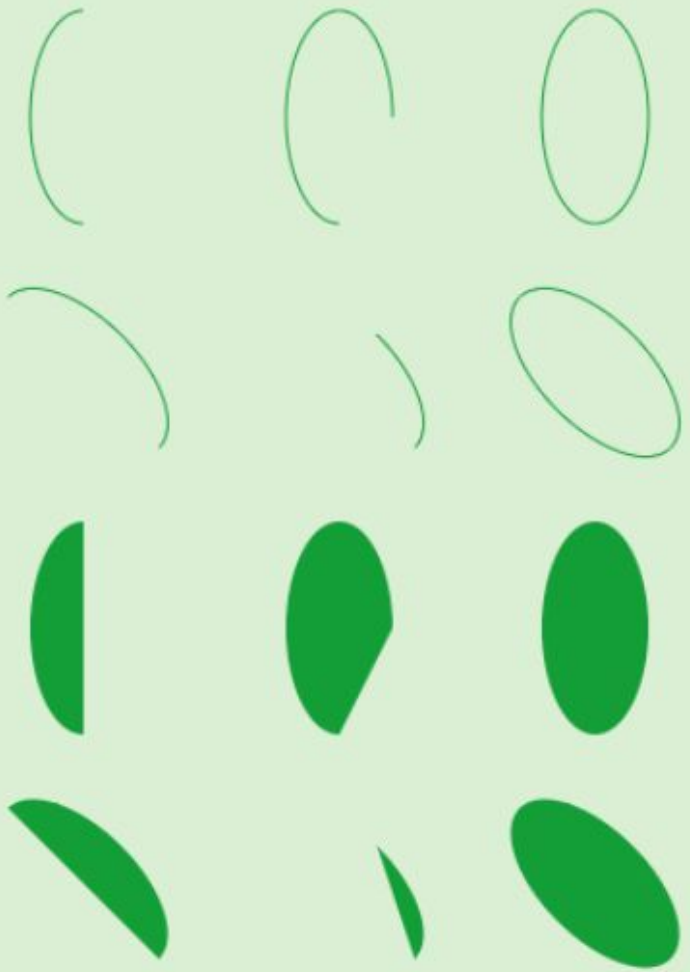
```
for (i=0;i<4;i++){
 for(j=0;j<3;j++){
  ctx.beginPath();
  var x           = 150+j*120;
  var y           = 150+i*120;
  var radius      = 50;
  var startAngle  = 0;
  var endAngle    = Math.PI+(Math.PI*j)/2;
  var anticlockwise  = i%2==1;
 ctx.arc(x,y,radius,startAngle,endAngle, anticlockwise);
    if (i>1){
   ctx.fill();
  } else {
   ctx.stroke();
  }  } }
```



# Эллипс

```
ctx.strokeStyle = '#139e38'; ctx.fillStyle = '#139e38';
for (i=0;i<4;i++)
        {   for(j=0;j<3;j++){
    ctx.beginPath();
var x   = 150+j*120;  var y = 150+i*120;
var radiusx        = 50;    var radiusy  = 25;
var rotation = Math.PI*0.5;  var startAngle     = 0;
var endAngle       = Math.PI+(Math.PI*j)/2;
var anticlockwise  = i%2==1;
     if (i%2==1) rotation = Math.PI*0.25;
ctx.ellipse(x,y,radiusx,radiusy,rotation,startAngle,end
Angle, anticlockwise);

      if (i>1){      ctx.fill();     }
        else {      ctx.stroke();     }
  }
}
```



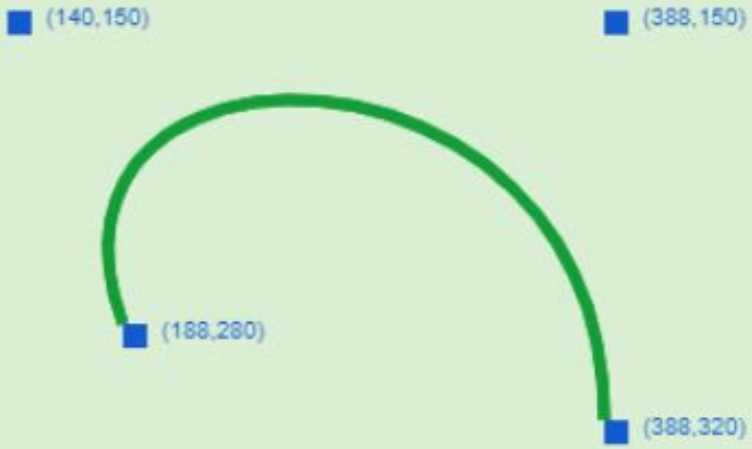
# Пример использования arcTo

```
const p0 = { x: 230, y: 150  }
const p1 = { x: 215,  y: 310 }
const p2 = { x: 60,  y: 180  }
ctx.beginPath();
ctx.moveTo(p0.x, p0.y);
ctx.arcTo(p1.x, p1.y, p2.x, p2.y, 30);
ctx.lineTo(p2.x, p2.y);
ctx.stroke();
```

# Кривая Безье-1


```
const p0 = { x: 188, y: 280  }
const p1 = { x: 140,  y: 150 }
const p2 = { x: 388,  y: 150  }
const p3 = { x: 388,  y: 320  }
ctx.moveTo(p0.x, p0.y);
    ctx.quadraticCurveTo(p1.x, p1.y, p3.x, p3.y);
    ctx.lineWidth = 10;
    ctx.strokeStyle = '#A0522D';
    ctx.stroke();
```

# Кривая Безье-2




```
const p0 = { x: 188, y: 280  }

const p1 = { x: 140,  y: 150 }

const p2 = { x: 388,  y: 150  }

const p3 = { x: 388,  y: 320  }

ctx.moveTo(p0.x, p0.y);

   ctx.bezierCurveTo(p1.x, p1.y, p2.x, p2.y, p3.x,
p3.y);

   ctx.lineWidth = 10;

   ctx.strokeStyle = '#A0522D';

   ctx.stroke();
```

# Стили линий

- lineWidth = value Устанавливает ширину линий, рисуемых в будущем.
- lineCap = type Устанавливает внешний вид концов линий.
- lineJoin = type Устанавливает внешний вид «углов», где встречаются линии.
- getLineDash() Возвращает текущий массив тире штриховки, содержащий четное число неотрицательных чисел.
- setLineDash(segments) Устанавливает текущий пунктир линии.
- lineDashOffset = value Указывает, где следует начинать тире массива в строке.

# Пример использования lineCap

```
var lineCap = ['butt','round','square'];
ctx.strokeStyle = 'black';
 for (var i=0;i<lineCap.length;i++){
   ctx.lineWidth = 15;
   ctx.lineCap = lineCap[i];
   ctx.beginPath();
   ctx.moveTo(25+i*50,10);
   ctx.lineTo(25+i*50,140);
   ctx.stroke();
}
```

# Пример использования lineJoin

```
var lineJoin = ['round','bevel','miter'];
  ctx.lineWidth = 10;
  for (var i=0;i<lineJoin.length;i++){
    ctx.lineJoin = lineJoin[i];
    ctx.beginPath();
    ctx.moveTo(-5,5+i*40);
    ctx.lineTo(35,45+i*40);
    ctx.lineTo(75,5+i*40);
    ctx.lineTo(115,45+i*40);
    ctx.lineTo(155,5+i*40);
    ctx.stroke();
  }
```

# Пример использования setLineDash(segments) и lineDashOffset = value

```
ctx.lineWidth = 10;
ctx.setLineDash([5,10,15,10]);
console.log(ctx.getLineDash());
ctx.beginPath();
ctx.moveTo(15, 15);
ctx.lineTo(15, 450);
ctx.stroke();
ctx.closePath();

ctx.beginPath();
ctx.strokeStyle = 'red';
```

# Заливки и границы фигур

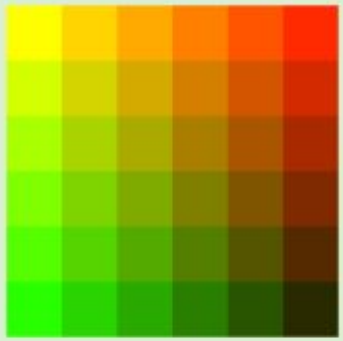
- fillStyle
- strokeStyle

```
ctx.fillStyle = "orange";
ctx.fillStyle = "#FFA500";
ctx.fillStyle = "rgb(255,165,0)";
ctx.fillStyle = "rgba(255,165,0,1)";
```

# Примеры fillStyle strokeStyle

```
for (var i=0;i<6;i++){
  for (var j=0;j<6;j++){
    ctx.fillStyle = 'rgb(' + Math.floor(255-42.5*i) + ',' +
                    Math.floor(255-42.5*j) + ',0)';
    ctx.fillRect(j*25+50,i*25+50,25,25);
  }  }

  for (var i=0;i<6;i++){
    for (var j=0;j<6;j++){
      ctx.strokeStyle = 'rgb(0,' + Math.floor(255-42.5*i) + ',' +
                      Math.floor(255-42.5*j) + ')';
      ctx.beginPath();
      ctx.lineWidth = 5;
      ctx.arc(12.5+j*25+250,12.5+i*25+50,10,0,Math.PI*2,true);
      ctx.stroke();
      ctx.closePath();
    }    }
```

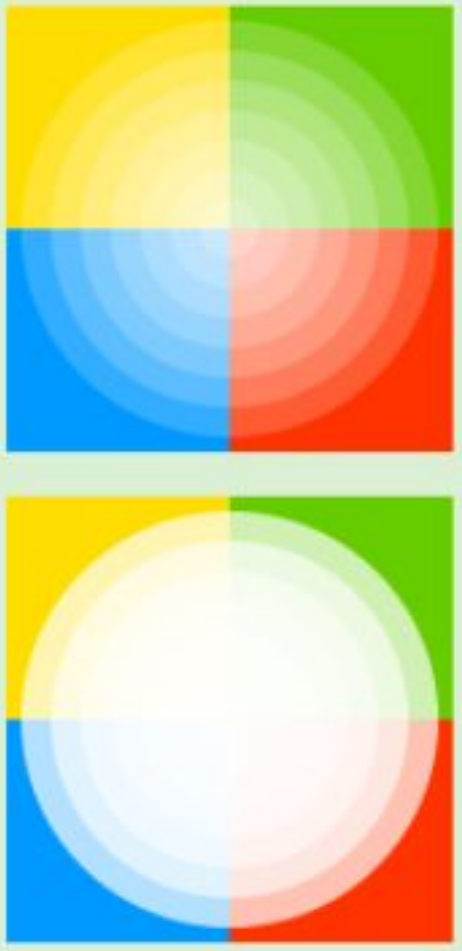


# Прозрачность

```
 ctx.fillStyle = '#FD0';    ctx.fillRect(0+x1,0+y1,75,75);
  ctx.fillStyle = '#6C0';     ctx.fillRect(75+x1,0+y1,75,75);
  ctx.fillStyle = '#09F';     ctx.fillRect(0+x1,75+y1,75,75);
  ctx.fillStyle = '#F30';    ctx.fillRect(75+x1,75+y1,75,75);
  ctx.fillStyle = '#FFF';   ctx.fillStyle = '#FFF’;
ctx.globalAlpha = 0.2;

 for (i=0;i<7;i++){
ctx.beginPath();     ctx.arc(75+x1,75+y1,10+10*i,0,Math.PI*2,true);
ctx.fill();   }

 ctx.globalAlpha = 1;
  for (i=0;i<7;i++){
    ctx.beginPath();
    ctx.fillStyle = 'rgba(255,255,255,'+(i+1)/10+')';
    ctx.arc(75+x1,75+y1,10+10*i,0,Math.PI*2,true);
    ctx.fill();  }
```



# Линейный градиент

```
var my_gradient = context.createLinearGradient(0, 0, 300, 0);
```

Поскольку значения у второго и четвертого параметра равно 0, этот градиент будет заполнен слева направо.

Градиент имеет два или более цвета остановки, которые могут быть в любом месте вдоль градиента. Чтобы добавить цвет остановки, необходимо указать его позицию вдоль градиента, она может быть от 0 до 1.

```
my_gradient.addColorStop(0, "black");
```

```
my_gradient.addColorStop(1, "white");
```

Третий цвет

```
my_gradient.addColorStop(0.5, '#ff0'); // желтый
```

Определение градиента не рисует что-либо на холсте, это просто объект, спрятанный где-то в памяти.

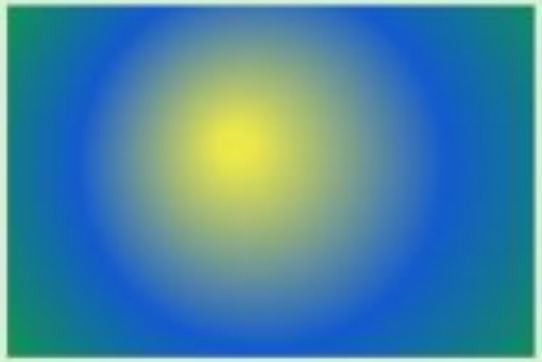
Для создания вертикального градиента необходимо оставить значения x (первый и третий параметр) постоянными и изменять значения y (второй и четвертый параметр) в диапазоне от 0 до высоты холста.

# Радиальный градиент

Для радиального градиента необходимо указать радиус внутренней и внешней окружности градиента. Координаты (x,y) определяют центры этих окружностей.

createRadialGradient(sx, sy, sr, dx, dy, dr)

```
var grd=ctx.createRadialGradient(75,150,5,90,160,100);
grd.addColorStop(0,'#DEB887');
grd.addColorStop(0.5,'#A0522D');
grd.addColorStop(1,'#A52A2A');
ctx.fillStyle=grd;
ctx.fillRect(10,110,150,100);
```



# Тени

shadowOffsetX = float ; shadowOffsetY = float  Горизонтальное и вертикальное  расстояние, на которое тень должна простираться от объекта.

shadowBlur = float Размер эффекта размытия

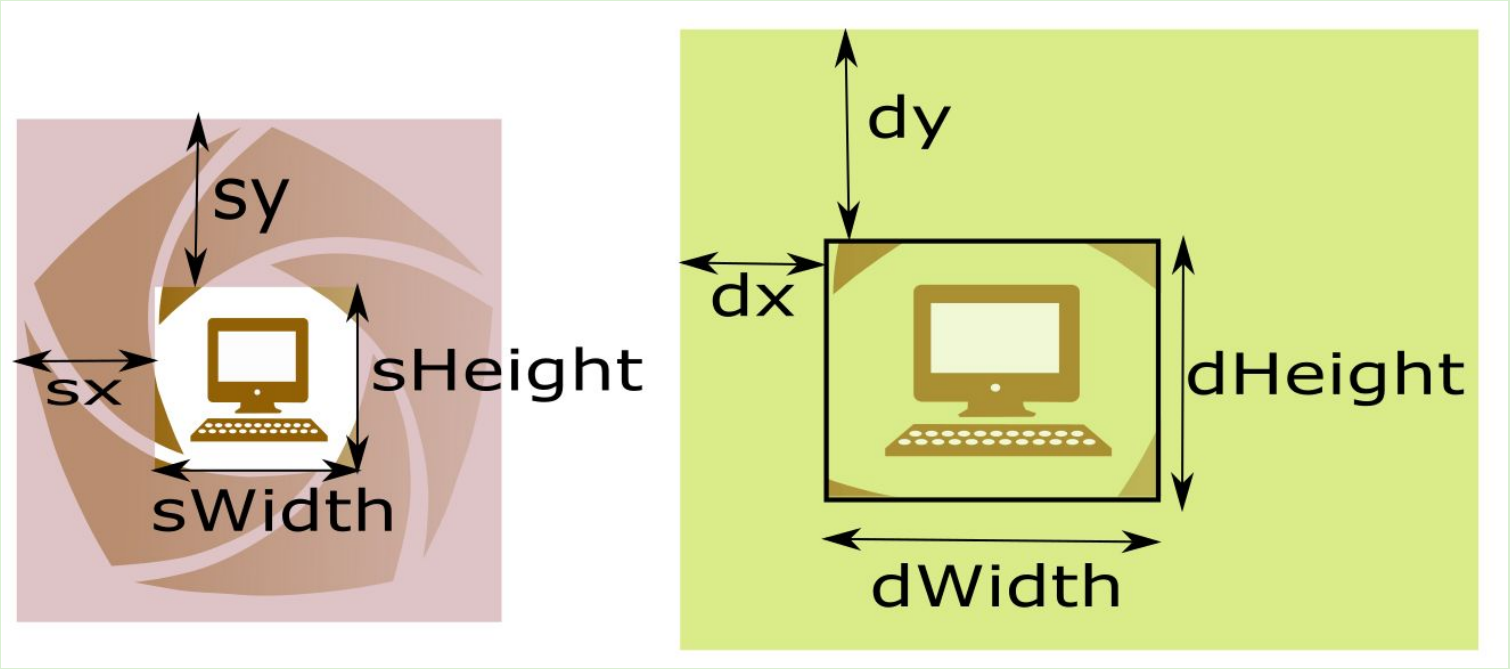
shadowColor = color Стандартное значение цвета CSS, указывающее цвет эффекта тени

```
ctx.shadowOffsetX = 2;
 ctx.shadowOffsetY = 2;
 ctx.shadowBlur = 2;
 ctx.shadowColor = "rgba(0, 0, 0, 0.5)";
  ctx.font = "20px Times New Roman";
 ctx.fillStyle = '#800000';
 ctx.fillText("Sample String", 5, 130);
```


```
ctx.shadowOffsetX = 5;
ctx.shadowOffsetY = 5;
ctx.shadowBlur = 4;
ctx.shadowColor = 'rgba(255, 0, 0, 0.5)';
ctx.fillStyle = '#8B4513';
ctx.fillRect(20, 180, 150, 100);
```

# Изображение

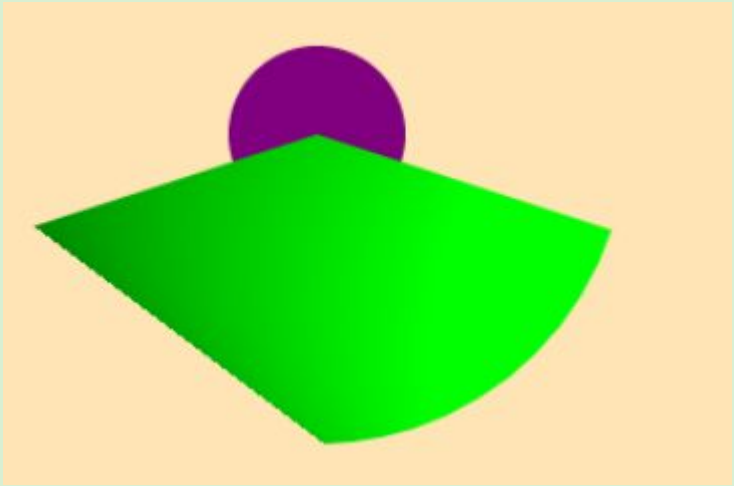
1. drawImage(image, dx, dy)
2. drawImage(image, dx, dy, dWidth, dHeight)
3. drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight)



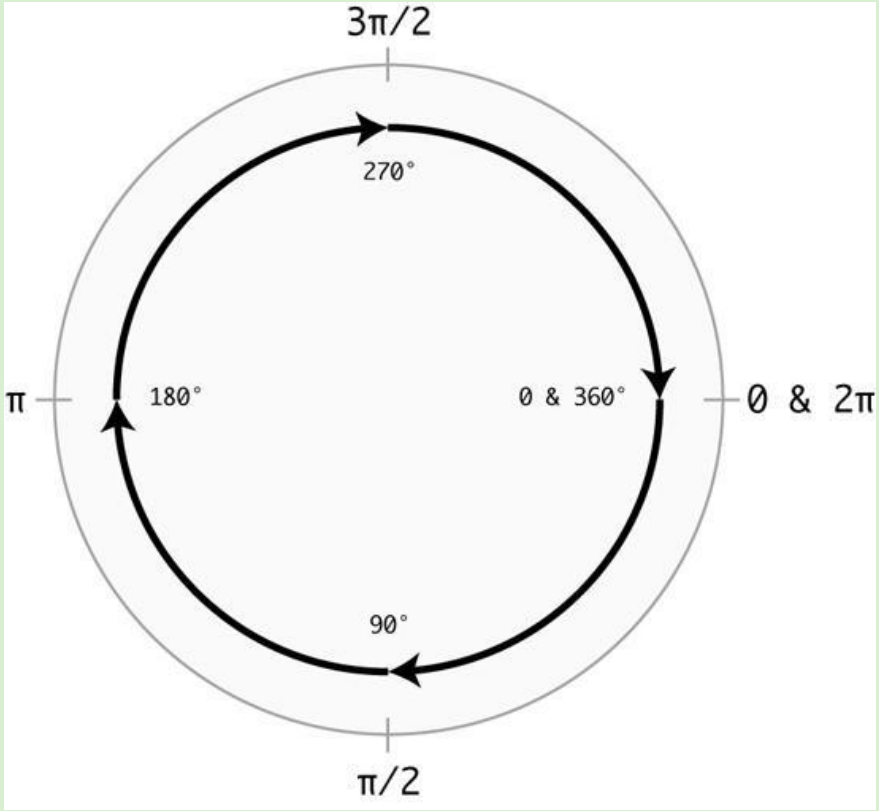
# Типичные ошибки

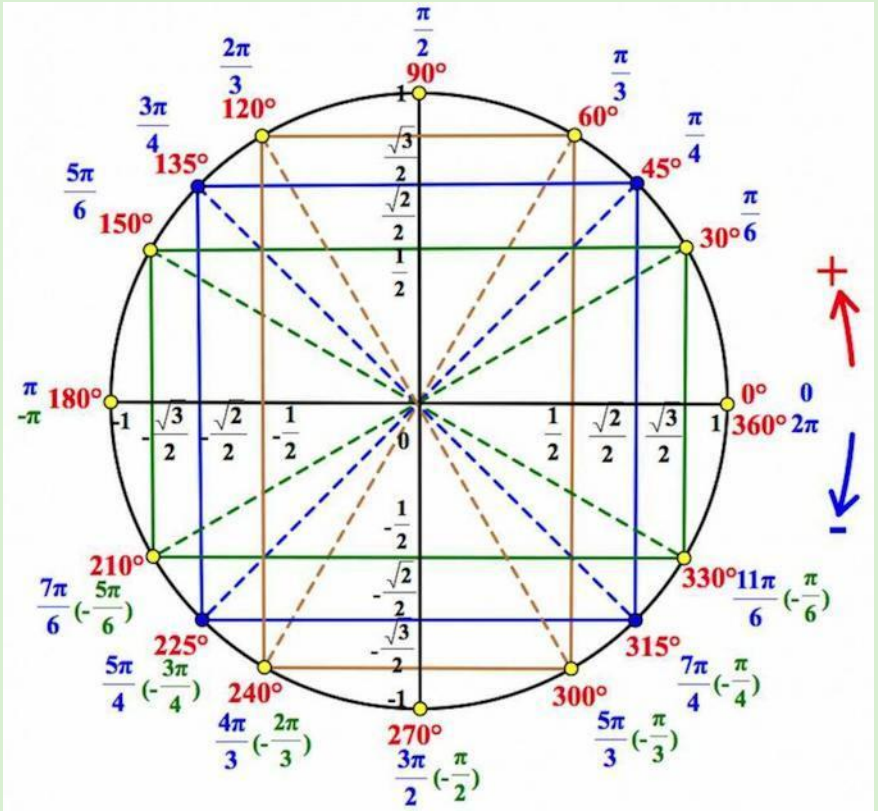
- Последовательность вывода объектов на экран
- Положение окружностей для задания градиента
- Синтаксис, наименования функций

```
ctx.beginPath();
ctx.fillStyle = '#800080';
ctx.arc(200, 300, 40, 0* Math.PI, 2 * Math.PI, true);
ctx.fill();
ctx.closePath();
var grd = ctx.createRadialGradient(100, 300, 50, 400,
400, 150);
    grd.addColorStop(0, '#008000');
    grd.addColorStop(1, '#00FF00');
    ctx.fillStyle = grd;
 ctx.beginPath();
ctx.arc(200, 300, 140, 0.9 * Math.PI, 0.1 * Math.PI, true);
ctx.lineTo(200,300);
ctx.fill();ctx.closePath();
```



# Углы в радианах



# Ресурсы

Таблица цветов

https://colorscheme.ru/html-colors.html

Канва on-line

https://www.w3schools.com/graphics/tryit.asp?filename=trycanvas_filltext

https://jsfiddle.net/jdias/ztpBF/

# Спасибо за внимание!