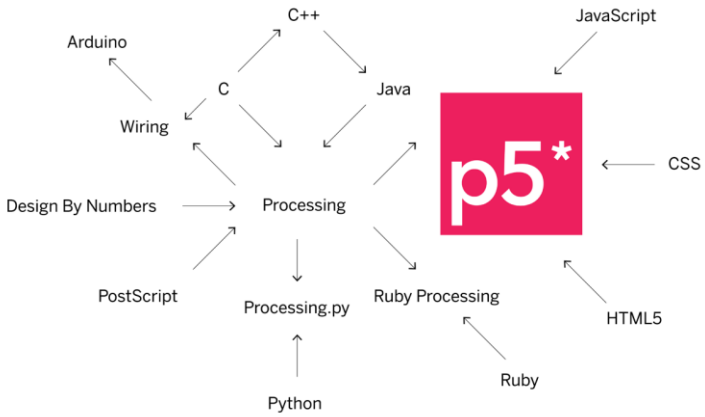
Графика в P5.js

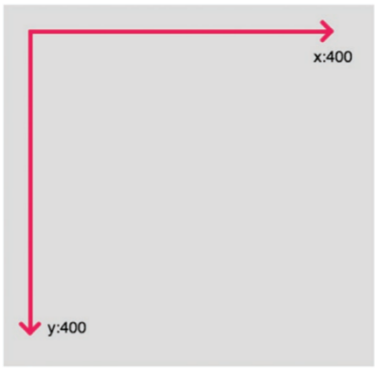
P5.js является бесплатным, и независимым от платформы, поэтому приложения могут работать в любой операционной системе, также p5.js имеет большое семейство языков программирования.



Фигура 1. Языки программирования связанные с p5.js

Как начать писать код в P5.js

Перед тем, как приступить к созданию собственных программ, необходимо знать, что любая фигура, созданная в среде p5.js, связана с системой координат, начало системы координат в любой программе — это крайний левый угол экрана. Вертикальная ось называется осью Y, а горизонтальная ось называется осью X. Увеличение значений координат x и y показано на рисунке 2.

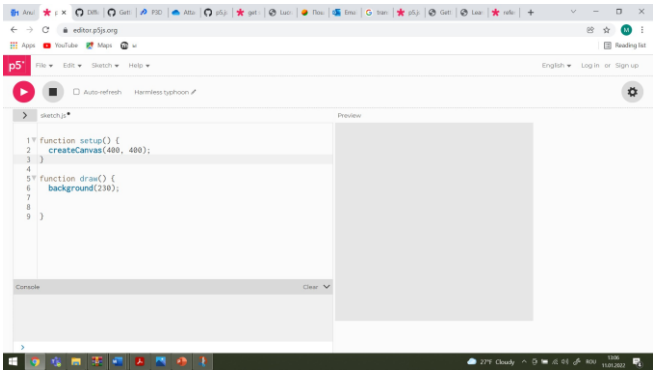


Фигура 2. Система координат в p5.js

Для создания простой программы на p5.js используем следующий шаблон:

```
function setup(){
    createCanvas(400,400);
}
function draw(){
    background(220);}
```

Результат работы программы и пример онлайн-редактора p5.js показан на рисунке 3.



Фигура 3. Рабочее окно в среде p5.js

Функция setup () вызывается один раз при запуске программы. Она используется для определения начальных свойств среды, таких как размер экрана и цвет фона, а также для загрузки мультимедийных файлов, таких как изображения и шрифты, при запуске программы. Для каждой программы может быть только одна функция setup (), и ее не следует вызывать после первоначального выполнения.

Примечание. Переменные, объявленные в setup (), не доступны в других функциях, включая draw ().

```
function setup() {
createCanvas();
}
```

Создает элемент canvas в документе и устанавливает его размеры в пикселях. Этот метод следует вызывать только один раз в начале установки. Вызов createCanvas более одного раза в эскизе приведет к очень непредсказуемому поведению. Если вам нужно более одного холста для рисования, вы можете использовать createGraphics (по умолчанию скрыто, но может отображаться).

Ширина и высота системных переменных задаются параметрами, передаваемыми этой функции. Если createCanvas () не используется, окну будет присвоен размер по умолчанию 100x100 пикселей.

```
createCanvas(w, h, [renderer]);
```

w число: ширина холста

h число: высота холста

Константа renderer: P2D или WEBGL (необязательно, WEBGL для 3D графики)

Функция draw () вызванный непосредственно после setup (), функция draw () непрерывно выполняет строки кода, содержащиеся в своем блоке, до

тех пор, пока программа не будет остановлена или не будет вызвана noLoop (). Обратите внимание, что если в setup () вызывается noLoop (), draw () все равно будет выполняться один раз перед остановкой. draw () вызывается автоматически и никогда не должен вызываться явно.

Он всегда должен управляться с помощью noLoop (), redraw () и loop (). После того, как noLoop () останавливает выполнение кода в draw (), redraw () приводит к тому, что код внутри draw () выполняется один раз, а loop () заставляет код внутри draw () возобновлять непрерывное выполнение.

Количество выполнений draw () в каждую секунду можно контролировать с помощью функции frameRate ().

Может быть только одна функция draw () для каждого эскиза, и draw () должна существовать, если вы хотите, чтобы код выполнялся непрерывно или обрабатывал события, такие как mousePressed (). Иногда в вашей программе может быть пустой вызов метода draw (), как показано в приведенном выше примере.

Важно отметить, что система координат рисования будет сбрасываться в начале каждого вызова draw (). Если какие-либо преобразования выполняются в draw () (например: масштабирование, поворот, перевод), их эффекты будут отменены в начале draw (), поэтому преобразования не будут накапливаться с течением времени.

```
function draw() {
---------
}
```

Функция background () устанавливает цвет, используемый для фона холста. Фон по умолчанию прозрачный. Эта функция обычно используется в draw () для очистки окна отображения в начале каждого кадра, но ее можно

использовать внутри setup (), чтобы установить фон для первого кадра анимации или если фон нужно установить только один раз.

Цвет указывается с точки зрения цвета RGB, HSB или HSL, в зависимости от текущего colorMode. (Цветовое пространство по умолчанию - RGB, каждое значение находится в диапазоне от 0 до 255). Диапазон альфа по умолчанию также составляет от 0 до 255.

Если указан один строковый аргумент, поддерживаются цветовые строки RGB, RGBA и Hex CSS и все именованные цветовые строки. В этом случае значение альфа-числа в качестве второго аргумента не поддерживается, следует использовать форму RGBA.

Объект Color также может быть предоставлен для установки цвета фона.

Изображение Image также может быть предоставлено для установки фонового изображения.

```
background(color);
background(colorstring, [a]);
background(gray, [a]);
background(v1, v2, v3, [a]);
background(values);
background(image, [a]);
```

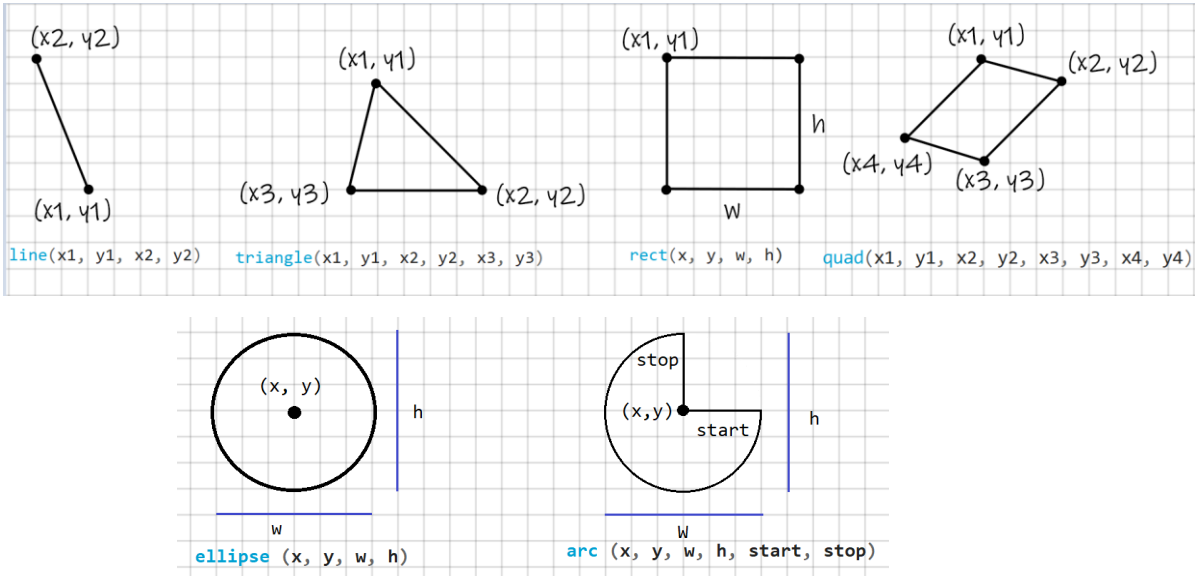
color:	любое значение, созданное функцией color ()
colorstring:	строка цвета, возможные форматы: целое число rgb () или rgba (), процентное отношение rgb () или rgba (), 3-значный шестнадцатеричный, 6-значный шестнадцатеричный
а число:	непрозрачность фона относительно текущего цветового диапазона (по умолчанию 0-255) (необязательно)

gray число:	указывает значение между белым и черным
v1 число:	красный или значение оттенка (в зависимости от текущего цветового режима)
v2 число:	значение зеленого или насыщенности (в зависимости от текущего цветового режима)
v3 число:	синий или значение яркости (в зависимости от текущего цветового режима)
values Number []:	массив, содержащий красный, зеленый, синий и альфа-компоненты цвета
image:	изображение, созданное с помощью loadImage () или createImage (), для установки в качестве фона (должно быть того же размера, что и окно эскиза)

Глава I

1.1 Создание простых 2D графических примитив

Простые графические примитивы — это геометрические фигуры, которые можно создавать с помощью функций в графической библиотеке P5.js. Простейшими графическими примитивами являются двумерные графические примитивы, на рис. 1.1 показано соответствие точек геометрических фигур и параметров, которые должны быть указаны в качестве аргументов функции в коде программы.



Фигура 1.1. Связь между геометрическими точками и параметрами функции P5.js

Функция arc (): рисует дугу на экране. Если вызывается только с x, y, w, h, start и stop, дуга будет нарисована и заполнена как открытый круговой сегмент. Если указан параметр режима, дуга будет заполнена как открытый полукруг (OPEN), замкнутый полукруг (CHORD) или как замкнутый круговой

сегмент (PIE). Источник может быть изменен с помощью функции ellipseMode (). Разница между режимами рисования показана на рисунке 1.3.

Дуга всегда рисуется по часовой стрелке от того места, где начинается падение, до того места, где остановка падает на эллипсе. Добавление или вычитание TWO_PI для любого угла не меняет места их падения. Если и старт, и остановка падают в одном и том же месте, будет нарисован полный эллипс. Имейте в виду, что ось Y увеличивается в направлении вниз, поэтому углы измеряются по часовой стрелке от положительного направления X («3 часа»).

Начальная точка и конечная точка могут использовать константы p5.js в соответствии со значениями, указанными в окружности, показанной на рисунке 1.2.

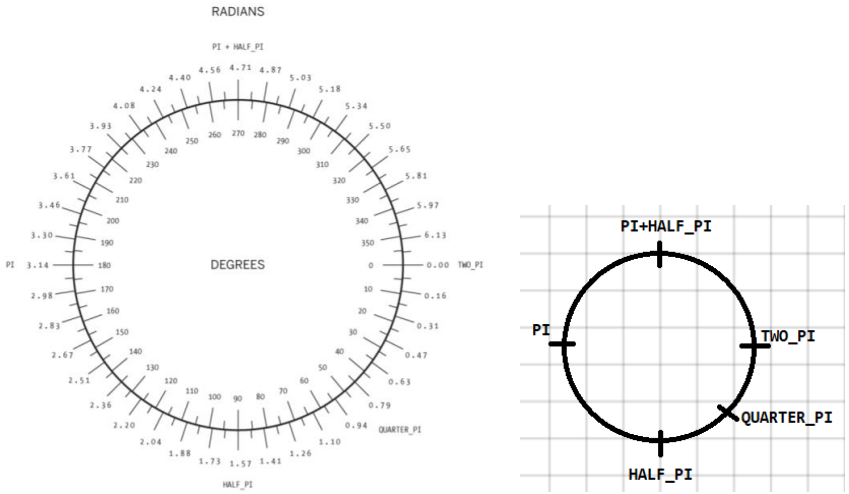


Figura 1.2. Relația dintre constante, radiani și grade în p5.js

```
arc (x, y, w, h, start, stop, [mode], [detail]);
```

x число: x-координата эллипса дуги

y число: y-координата эллипса дуги

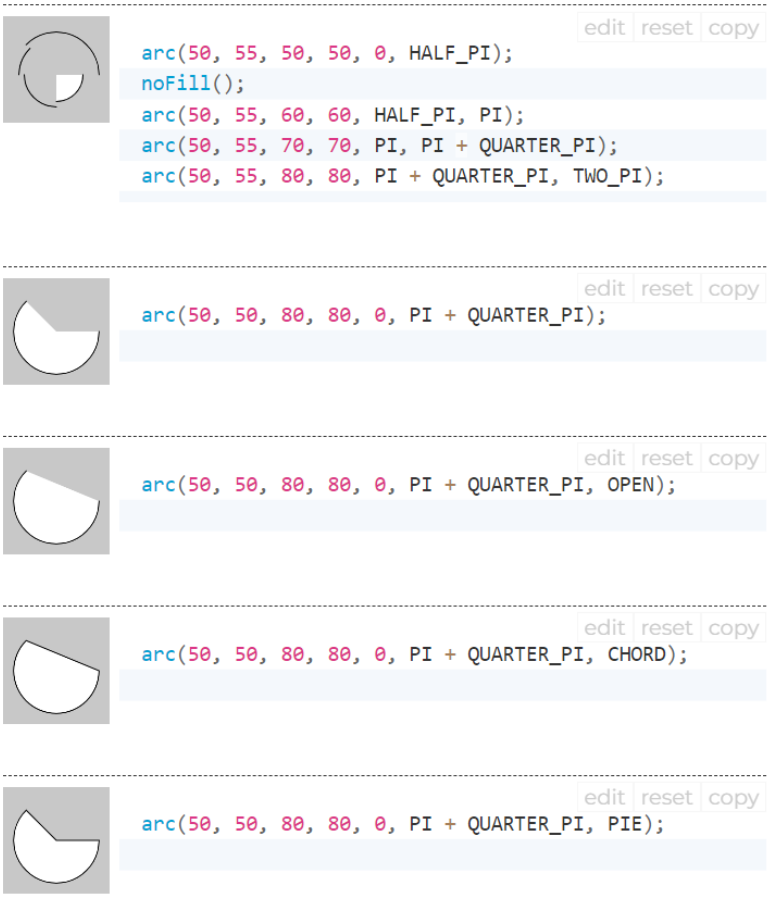
w число: ширина эллипса дуги по умолчанию

h число: высота эллипса дуги по умолчанию

start число: угол начала дуги, указанный в радианах

stop число: угол остановки дуги, указанный в радианах

mode константа: параметр для определения способа рисования дуги. CHORD, PIE или OPEN (необязательно)



Фигура 1.2. Примеры рисования дуги

Функция ellipse(): рисует эллипс (овал) на экране. Эллипс с равной шириной и высотой - это круг. По умолчанию первые два параметра задают местоположение, а третий и четвертый параметры определяют ширину и высоту фигуры. Если высота не указана, значение ширины используется как для ширины, так и для высоты. Если указана отрицательная высота или

ширина, берется абсолютное значение. Источник может быть изменен с помощью функции ellipseMode ().

Синтаксис

```
ellipse(x, y, w, [h]);
ellipse (x, y, w, h, detail);
```

параметры

x Число: x-координата эллипса.

y Число: y-координата эллипса.

w Число: ширина эллипса.

h Число: высота эллипса. (Необязательный)

detail Целое число: количество радиальных секторов для рисования (для режима WebGL)

Функция circle(): рисует круг на экране. Эта функция является частным случаем функции ellipse (), где ширина и высота эллипса одинаковы. Высота и ширина эллипса соответствуют диаметру круга. По умолчанию первые два параметра задают расположение центра круга, третий - диаметр круга.

Синтаксис

```
circle(x, y, d);
```

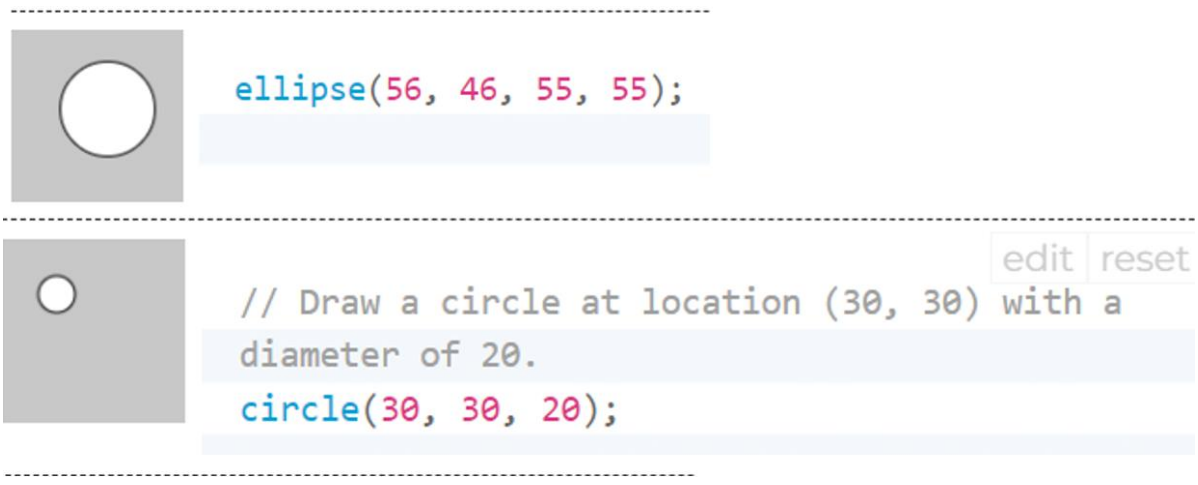
Параметры

x Число: x-координата центра круга.

y Число: y-координата центра круга.

d Число: диаметр круга.

На рис. 1.3 показан пример функций ellipse и circle.



Фигура 1.3. Примеры функций circle и ellips

Функция line(): рисует линию (прямой путь между двумя точками) к экрану. Версия line () с четырьмя параметрами рисует линию в 2D. Чтобы закрасить линию, используйте функцию stroke(). Строка не может быть заполнена, поэтому функция fill () не повлияет на цвет строки. 2D линии по умолчанию рисуются с шириной в один пиксель, но это можно изменить с помощью функции strokeWeight ().

Синтаксис

```
line(x1, y1, x2, y2);
line(x1, y1, z1, x2, y2, z2);
```

Параметры

x1: x-координата первой точки

y1: y-координата первой точки

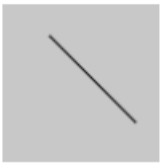
x2: x-координата второй точки

y2: y-координата второй точки

z1: координата z первой точки

z2: координата z второй точки

Пример:



```
line(30, 20, 85, 75);
```

Функция point(): рисует точку, координату в пространстве размером один пиксель. Первый параметр - это горизонтальное значение для точки, второй - вертикальное значение для точки. Цвет точки изменяется с помощью функции stroke(). Размер точки изменяется с помощью функции strokeWeight().

Синтаксис

```
point(x, y, [z]);
point(coordinate_vector);
```

Параметры

x: x-координата

y: y-координата

z: координата z (для режима WebGL) (необязательно)

coordonate_vector: вектор координат;

Примеры программы функции point показаны на рис. 1.4.

Фигура 1.4. Примеры программы функции point

Функция quad(): рисует четырехугольник. Четырехугольник - четырехсторонний многоугольник. Он похож на прямоугольник, но углы между его краями не ограничены до девяноста градусов. Первая пара параметров (x1, y1) задает первую вершину, а последующие пары должны двигаться по часовой стрелке или против часовой стрелки вокруг определенной формы. z-аргументы работают только тогда, когда quad () используется в режиме WEBGL.

Синтаксис

```
quad(x1, y1, x2, y2, x3, y3, x4, y4);
quad(x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4);
```

Параметры

x1: x-координата первой точки

y1: y-координата первой точки

x2: x-координата второй точки

y2: y-координата второй точки

x3: x-координата третьей точки

y3: y-координата третьей точки

x4: x-координата четвертой точки

y4: y-координата четвертой точки

z1: координата z первой точки

z2: координата z второй точки

z3: координата z третьей точки

z4: координата z четвертой точки

Функция rect (): рисует прямоугольник на экране. Прямоугольник - это четырехсторонняя фигура с каждым углом в девяносто градусов. По умолчанию первые два параметра задают расположение верхнего левого угла, третий - ширину, а четвертый - высоту. Однако способ интерпретации этих параметров можно изменить с помощью функции rectMode ().

Пятый, шестой, седьмой и восьмой параметры, если указаны, определяют радиус угла для верхнего левого, верхнего правого, нижнего правого и нижнего левого углов соответственно. Опущенный параметр радиуса угла устанавливается равным значению ранее указанного значения радиуса в списке параметров.

Синтаксис

```
rect(x, y, w, h, [tl], [tr], [br], [bl]);
rect(x, y, w, h, [detailX], [detailY]);
```

Параметры

x: x-координата прямоугольника.

y: y-координата прямоугольника.

w r: ширина прямоугольника.

h: высота прямоугольника.

tl: радиус верхнего левого угла. (Необязательный)

tr: радиус верхнего правого угла. (Необязательный)

br: радиус нижнего правого угла. (Необязательный)

bl: радиус нижнего левого угла. (Необязательный)

detailX Integer: количество сегментов в направлении x (для режима WebGL) (необязательно)

detailY Integer: количество сегментов в направлении y (для режима WebGL) (необязательно)

Примеры программы и их результаты показаны на рис. 1.5.

Фигура 1.5. Примеры программ функций rect şi quad

Функция square(): рисует квадрат на экране. Квадрат - это четырехсторонняя форма с каждым углом в девяносто градусов и равным размером стороны. Эта функция является частным случаем функции rect (), где ширина и высота одинаковы, а параметр называется «s» для размера стороны. По умолчанию первые два параметра задают расположение верхнего левого угла, третий - размер стороны квадрата. Однако способ интерпретации этих параметров можно изменить с помощью функции rectMode ().

Четвертый, пятый, шестой и седьмой параметры, если они указаны, определяют радиус угла для левого верхнего, правого верхнего, правого нижнего и левого нижнего углов соответственно. Опущенный параметр

радиуса угла устанавливается равным значению ранее указанного значения радиуса в списке параметров.

Синтаксис

```
square(x, y, s, [tl], [tr], [br], [bl]);
```

Параметры

x: x-координата квадрата.

y: y-координата квадрата.

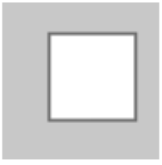
s: размер стороны квадрата.

tl: радиус верхнего левого угла. (Необязательный)

tr: радиус верхнего правого угла. (Необязательный)

br: радиус нижнего правого угла. (Необязательный)

bl: радиус нижнего левого угла. (Необязательный)



edit reset copy

```
// Draw a square at location (30, 20) with a side
size of 55.
square(30, 20, 55);
```

edit reset copy

```
// Draw a square with rounded corners, each having
a radius of 20.
square(30, 20, 55, 20);
```

edit reset copy

```
// Draw a square with rounded corners having the
following radii:
// top-left = 20, top-right = 15, bottom-right =
10, bottom-left = 5.
square(30, 20, 55, 20, 15, 10, 5);
```

Фигура 1.6. Примеры программ функции square

Функция triangle(): Треугольник - это плоскость, созданная путем соединения трех точек. Первые два аргумента указывают первую точку, средние два аргумента указывают вторую точку, а последние два аргумента указывают третью точку.

Синтаксис

```
triangle(x1, y1, x2, y2, x3, y3);
```

Параметры

x1: x-координата первой точки

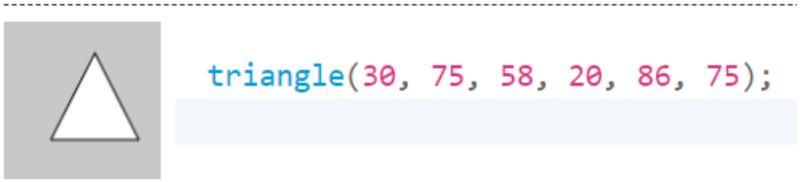
y1: y-координата первой точки

x2: x-координата второй точки

y2: y-координата второй точки

x3: x-координата третьей точки

y3: y-координата третьей точки



Фигура 1.7. Примеры программ функции triangle

1.2 Простые графические атрибуты

Особенности геометрических фигур, такие как цвет фигуры, толщина и цвет линии, тип штриховки, способ соединения линий, являются простыми графическими атрибутами. В библиотеке p5.js есть список функций, позволяющих устанавливать или изменять эти атрибуты.

Основные графические атрибуты описаны ниже:

Функция fill(): Устанавливает цвет, используемый для заливки фигур. Например, если вы запустите заливку (204, 102, 0), все фигуры, нарисованные после команды заливки, будут заполнены оранжевым цветом. Этот цвет указывается с точки зрения цвета RGB или HSB в зависимости от текущего colorMode (). (Цветовое пространство по умолчанию - RGB, каждое значение находится в диапазоне от 0 до 255). Диапазон альфа по умолчанию также составляет от 0 до 255.

Если указан один строковый аргумент, поддерживаются цветовые строки RGB, RGBA и Hex CSS и все именованные цветовые строки. В этом случае значение альфа-числа в качестве второго аргумента не поддерживается, следует использовать форму RGBA.

```
fill(v1, v2, v3, [alpha]);
fill(value);
fill(gray, [alpha]);
fill(values);
fill(color);
```

v1 число: красный или значение оттенка относительно текущего цветового диапазона

v2 число: значение зеленого или насыщенности относительно текущего цветового диапазона

v3 число: синий или значение яркости относительно текущего цветового диапазона

альфа число: (необязательно)

String: string цветная строка

серый номер: значение серого

значения Number []: массив, содержащий красный, зеленый, синий и альфа-компоненты цвета

```
// Grayscale integer value
fill(51);
rect(20, 20, 60, 60);
```

```
// R, G & B integer values
fill(255, 204, 0);
rect(20, 20, 60, 60);
```

```
// H, S & B integer values
colorMode(HSB);
fill(255, 204, 100);
rect(20, 20, 60, 60);
```

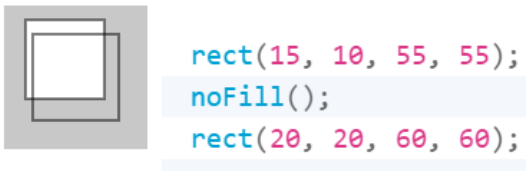
```
// Named SVG/CSS color string
fill('red');
rect(20, 20, 60, 60);
```

```
// three-digit hexadecimal RGB notation
fill('#fae');
rect(20, 20, 60, 60);
```

```
// six-digit hexadecimal RGB notation
fill('#222222');
rect(20, 20, 60, 60);
```

Фигура 1.8.а) Примеры использования функции fill

Чтобы фигура была прозрачной, используем функцию **nofill**.



```
rect(15, 10, 55, 55);
noFill();
rect(20, 20, 60, 60);
```

Фигура 1.8.б) Примеры использования функции nofill

Функция stroke ():

Задает цвет, используемый для рисования линий и границ фигур. Этот цвет указывается в терминах цвета RGB или HSB в зависимости от текущего colorMode () (цветовое пространство по умолчанию - RGB, каждое значение находится в диапазоне от 0 до 255). Альфа-диапазон по умолчанию также составляет от 0 до 255. Если указан единственный строковый аргумент, поддерживаются цветовые строки RGB, RGBA и Hex CSS и все именованные цветовые строки. В этом случае значение альфа-числа в качестве второго аргумента не поддерживается, следует использовать форму RGBA.

Синтаксис:

```
stroke(v1, v2, v3, [alpha]);
stroke(value);
stroke(gray, [alpha]);
stroke(values);
stroke(color);
```

Параметры

v1 - значение красного или оттенка относительно текущего цветового диапазона

v2 - зеленый цвет или значение насыщенности относительно текущего цветового диапазона

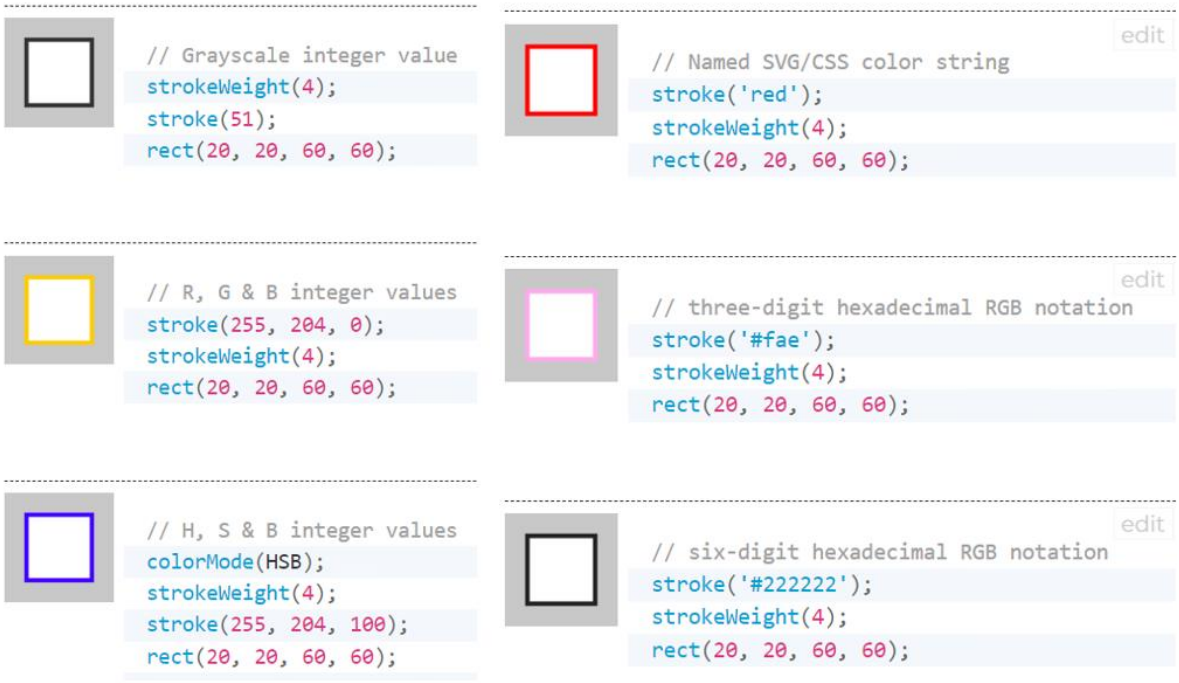
v3 - синий или значение яркости относительно текущего цветового диапазона

alpha -число: (необязательно)

value - цветовая строка

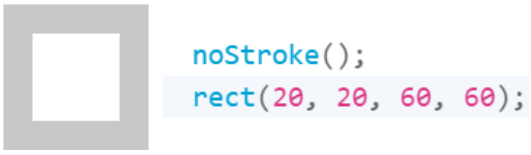
grey – число, значение серого

values число []: массив, содержащий красный, зеленый, синий и альфа-компоненты цвета



Фигура 1.9.а) Примеры использования функции stroke

Чтобы фигура рисовалась без рамки, используем функцию noStroke.



Фигура 1.9.б) Примеры использования функции noStroke

Помимо цвета, функция **stroke** может указывать следующие параметры:

strokeCap();

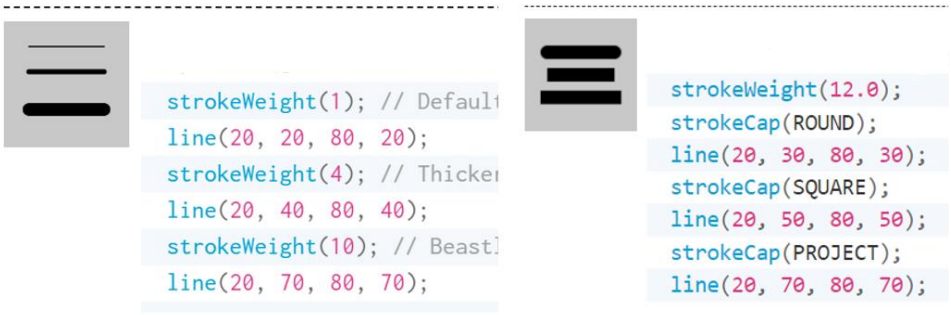
strokeJoin();

strokeWeight();

Функция strokeWeight(number): Устанавливает ширину линии. Все значения указаны в пикселях.

Функция strokeCap(cap): Устанавливает стиль воспроизведения конца строки. Эти концы либо закруглены, либо квадратны, либо расширены, каждый из которых указан с соответствующими параметрами: ROUND, SQUARE și PROIECT.

Примеры использования этих аргументов показаны на рис. 1. 10.



Фигура 1.10. Примеры использования функции stroke

Функция strokeJoin(join): Задает стиль соединения (стыковки) отрезков. Они могут быть указаны с соответствующими параметрами MITRE, BEVEL и ROUND. По умолчанию установлено значение MITRE.

Разница между этими режимами показана на рисунке 1.11.

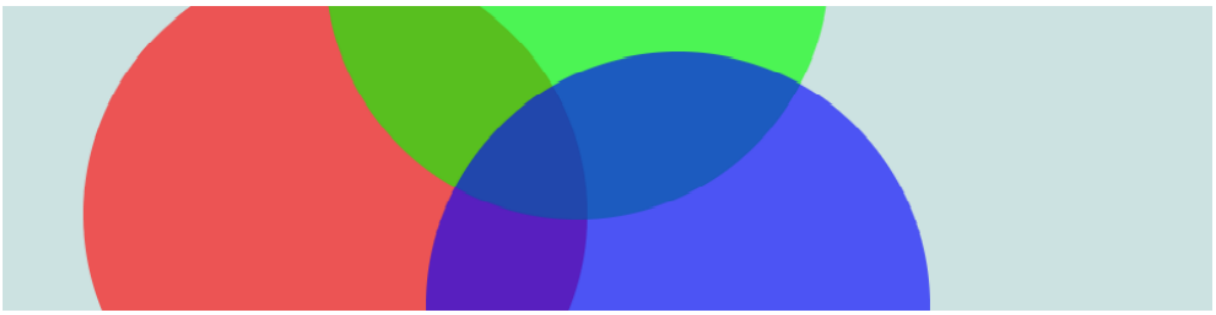
Фигура 1.11. Примеры использования функции strokeJoin

Установка прозрачности

Добавляя необязательный четвертый параметр к **fill()** или **stroke(),** известный как альфа-значение, можно контролировать прозрачность. Этот четвертый параметр использует диапазон от 0 до 255 устанавливает степень прозрачности. Значение 0 определяет цвет как полностью прозрачным (не отображается), значение 255 полностью непрозрачный, и значения между этими крайними значениями вызывают смешивание цветов на экране:

Пример:

```
function setup() {
 createCanvas(480, 120);
 noStroke();
}
function draw() {
 background(204, 226, 225);     // светло голубой цвет
 fill(255, 0, 0, 160);          // красный цвет
 ellipse(132, 82, 200, 200);    // красный круг
 fill(0, 255, 0, 160);          // зелёный цвет
 ellipse(228, -16, 200, 200);   // зелёный круг
 fill(0, 0, 255, 160);          // синий цвет
 ellipse(268, 118, 200, 200);   // синий круг
}
```



Если необходимо добавить текст, можно использовать текстовую функцию, которая может иметь разные параметры.

Функция text(): Рисует текст на экране, выводит информацию, указанную в первом параметре, на экран в позиции, заданной дополнительными параметрами. Будет использоваться шрифт по умолчанию, если шрифт не установлен с помощью функции textFont(), и размер по умолчанию будет использоваться, если шрифт с textSize() не установлен. Цвет текста можно изменить с помощью функции fill(). Изменение контура текста осуществляется с помощью функций stroke() и strokeWeight().

Текст отображается в соответствии с функцией textAlign(), которая предоставляет возможность рисовать левые, правые и центральные координаты.

Синтаксис :

text(str, x, y, [x2], [y2]);

Параметры:

str : указывает строку для отображения на экране;

x: координата x начальной точки текста;

y: координата y начальной точки текста;

x2 и y2: определяют прямоугольную область для отображения и могут использоваться только со строками. Когда эти параметры указаны, они интерпретируются на основе текущей настройки rectMode(). Текст, который не полностью помещается в указанный прямоугольник, не будет отображаться на экране. Если x2 и y2 не указаны, базовое выравнивание используется по умолчанию, что означает, что текст будет составлен из x и y.

Основные атрибуты текста приведены на рисунок. 1.12

Фигура 1.12. Примеры использования функций работы с текстом

Помимо основных фигур можно создавать и новые более сложные фигуры путём соединения ряда точек. Для этого используются функций **beginShape()** и **endShape().**

Функция **beginShape()** начинает определение вершин фигуры, а **endShape()** указывает что определение вершин закончено.

Синтаксис:

beginShape([kind]);

Значение параметра **kind** является опциональным и указывает, какие типы фигур следует создавать из указанных вершин.

Параметр **kind** может использовать следующие константы

POINTS – рисует серию точек;

LINES – рисует серию несоединённых отрезков (отдельных линий);

TRIANGLES – рисует серию отдельных треугольников;

TRIANGLE_FAN – рисует серию соединенных треугольников, разделяющих первую вершину веерообразно;

TRIANGLE_STRIP – рисует серию соединенных треугольников в стиле полос;

QUADS – рисует серию отдельных четырехугольников;

QUAD_STRIP – рисует четырехугольную полосу, используя соседние ребра, чтобы сформировать следующий четырехугольник;

TESS – рисует неправильный многоугольника

Без указания режима форма может быть любым неправильным многоугольником.

После вызова функции beginShape() должна следовать серия команд **vertex()**. Чтобы прекратить рисование фигуры, вызовите endShape(). Каждая фигура будет обведена текущим цветом обводки и заполнена цветом заливки.

Такие преобразования, как **translate()**, **rotate()** и **scale()** не работают в beginShape(). Также невозможно использовать другие формы, такие как ellipse() или rect() внутри beginShape().

Функция **vertex()** используется исключительно внутри функций beginShape() и endShape() для указания координат вершин точек. Все фигуры строятся путем соединения ряда вершин.

Синтаксис:

```
vertex(x, y);
vertex(x, y, [z]);
vertex(x, y, [z], [u], [v]);
```

Параметры:

`x` – x-координата вершины

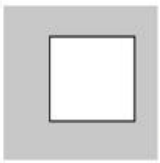
`y` – y-координата вершины

`z` – z-координата вершины, по умолчанию равна 0 если не указана

`u` – число u-координата текстуры вершины (необязательно)

v – число v-координата текстуры вершины (необязательно)

Ниже приведены примеры использования разных режимов параметра **kind**.



```
beginShape();
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape(CLOSE);
```

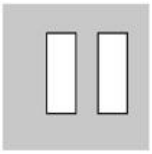
```
strokeWeight(3);
beginShape(POINTS);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```

```
beginShape(LINES);
vertex(30, 20);
vertex(85, 20);
vertex(85, 75);
vertex(30, 75);
endShape();
```

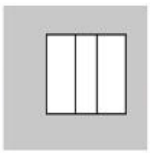
```
beginShape(TRIANGLE_STRIP);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
vertex(90, 75);
endShape();
```

```
beginShape(TRIANGLES);
vertex(30, 75);
vertex(40, 20);
vertex(50, 75);
vertex(60, 20);
vertex(70, 75);
vertex(80, 20);
endShape();
```

```
beginShape(TRIANGLE_FAN);
vertex(57.5, 50);
vertex(57.5, 15);
vertex(92, 50);
vertex(57.5, 85);
vertex(22, 50);
vertex(57.5, 15);
endShape();
```



```
beginShape(QUADS);
vertex(30, 20);
vertex(30, 75);
vertex(50, 75);
vertex(50, 20);
vertex(65, 20);
vertex(65, 75);
vertex(85, 75);
vertex(85, 20);
endShape();
```



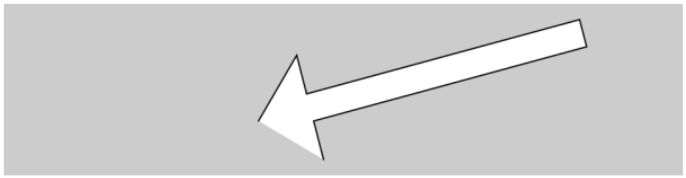
```
beginShape(QUAD_STRIP);
vertex(30, 20);
vertex(30, 75);
vertex(50, 20);
vertex(50, 75);
vertex(65, 20);
vertex(65, 75);
vertex(85, 20);
vertex(85, 75);
endShape();
```

```
beginShape(TESS);
vertex(20, 20);
vertex(80, 20);
vertex(80, 40);
vertex(40, 40);
vertex(40, 60);
vertex(80, 60);
vertex(80, 80);
vertex(20, 80);
endShape(CLOSE);
```

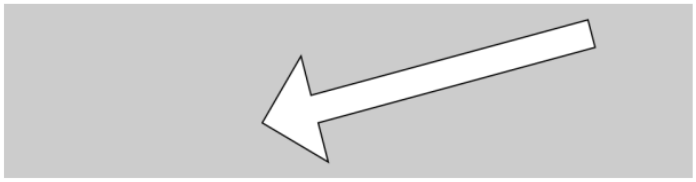
```
createCanvas(100, 100, WEBGL);
background(240, 240, 240);
fill(237, 34, 93);
noStroke();
beginShape();
vertex(-10, 10);
vertex(0, 35);
vertex(10, 10);
vertex(35, 0);
vertex(10, -8);
vertex(0, -35);
vertex(-10, -8);
vertex(-35, 0);
endShape();
```

Функция **endShape()** является дополнением к **beginShape()** и может вызываться только после **beginShape()**. При вызове endShape() все данные изображения, определенные с момента предыдущего вызова beginShape(), записываются в буфер изображения. Константа CLOSE в качестве значения параметра режима для закрытия формы (для соединения начала и конца).

Пример:



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape();
}
```



```
function setup() {
  createCanvas(480, 120);
}

function draw() {
  background(204);
  beginShape();
  vertex(180, 82);
  vertex(207, 36);
  vertex(214, 63);
  vertex(407, 11);
  vertex(412, 30);
  vertex(219, 82);
  vertex(226, 109);
  endShape(CLOSE);
}
```

Глава II

2.1 Создание простых 3D графических примитив

Основными графическими примитивами являются:

- plane()
- box()
- sphere()
- cylinder()
- cone()
- ellipsoid()
- torus()

Функция plane(): рисует плоскость с заданной шириной и высотой

Синтаксис

```
plane([width], [height], [detailX], [detailY]);
```

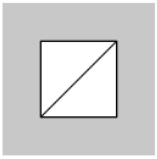
Параметры

[width]: ширина плоскости (необязательно)

[height]: высота самолета (необязательно)

detailX: количество треугольников в x-измерении (необязательно)

detailY: количество подразделений треугольника в y-измерении (необязательно)



edit reset

```
// draw a plane
// with width 50 and height 50
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a white plane with black wireframe
lines');
}

function draw() {
  background(200);
  plane(50, 50);
}
```

Функция box(): рисует прямоугольник с заданной шириной, высотой и глубиной

Синтаксис

```
box([width], [Height], [depth], [detailX], [detailY]);
```

Параметры

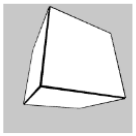
[width]: ширина коробки (необязательно)

[Height]: высота коробки (опционально)

[depth]: глубина коробки (необязательно)

detailX: количество треугольников в x-измерении (необязательно)

detailY: количество подразделений треугольника в y-измерении (необязательно)



edit reset

```
// draw a spinning box
// with width, height and depth of 50
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a white box rotating in 3D space');
}

function draw() {
  background(200);
  rotateX(frameCount * 0.01);
  rotateY(frameCount * 0.01);
  box(50);
}
```

Функция sphere(): рисует сферу с заданным радиусом.

DetailX и detailY определяют количество подразделений в x-измерении и y-измерении сферы. Больше подразделений делает сферу более гладкой. Рекомендуемые максимальные значения равны 24. Использование значения больше 24 может замедлить работу браузера.

Синтаксис

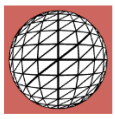
```
sphere([radius], [detailX], [detailY]);
```

Параметры

radius: радиус круга (необязательно)

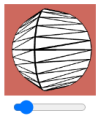
detailX: количество подразделений в x-измерении (необязательно)

detailY: количество подразделений в y-измерении (необязательно)



edit reset

```
// draw a sphere with radius 40
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a white sphere with black wireframe lines');
}

function draw() {
  background(205, 102, 94);
  sphere(40);
}
```



edit reset copy

```
let detailX;
// slide to see how detailX works
function setup() {
  createCanvas(100, 100, WEBGL);
  detailX = createSlider(3, 24, 3);
  detailX.position(10, height + 5);
  detailX.style('width', '80px');
  describe(
    'a white sphere with low detail on the x-axis, including a slider to adjust detailX'
  );
}

function draw() {
  background(205, 105, 94);
  rotateY(millis() / 1000);
  sphere(40, detailX.value(), 16);
}
```

edit reset copy

```
let detailY;
// slide to see how detailY works
function setup() {
  createCanvas(100, 100, WEBGL);
  detailY = createSlider(3, 16, 3);
  detailY.position(10, height + 5);
  detailY.style('width', '80px');
  describe(
    'a white sphere with low detail on the y-axis, including a slider to adjust detailY'
  );
}

function draw() {
  background(205, 105, 94);
  rotateY(millis() / 1000);
  sphere(40, 16, detailY.value());
}
```

Функция cylinder():рисует цилиндр с заданным радиусом и высотой

DetailX и detailY определяют количество подразделений в x-измерении и y-измерении цилиндра. Больше подразделений заставляет цилиндр казаться более гладким. Рекомендуемое максимальное значение для detailX равно 24. Использование значения больше 24 может вызвать предупреждение или замедлить работу браузера.

Синтаксис

```
cylinder([radius], [height], [detailX], [detailY], [bottomCap], [topCap]);
```

Параметры

radius: радиус поверхности (необязательно)

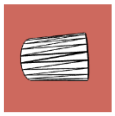
height: высота цилиндра (опция)

detailX: количество подразделений в x-измерении; по умолчанию 24 (необязательно)

detailY: количество подразделений в y-измерении; по умолчанию 1 (необязательно)

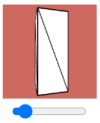
bottomCap Boolean: рисовать ли дно цилиндра (необязательно)

topCap Boolean: рисовать ли верх цилиндра (необязательно)



edit

```
// draw a spinning cylinder
// with radius 20 and height 50
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a rotating white cylinder');
}

function draw() {
  background(205, 105, 94);
  rotateX(frameCount * 0.01);
  rotateZ(frameCount * 0.01);
  cylinder(20, 50);
}
```



edit reset

```
// slide to see how detailX works
let detailX;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailX = createSlider(3, 24, 3);
  detailX.position(10, height + 5);
  detailX.style('width', '80px');
  describe(
    'a rotating white cylinder with limited X
detail, with a slider that adjusts detailX'
  );
}

function draw() {
  background(205, 105, 94);
  rotateY(millis() / 1000);
  cylinder(20, 75, detailX.value(), 1);
}
```

edit | reset

```
// slide to see how detailY works
let detailY;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailY = createSlider(1, 16, 1);
  detailY.position(10, height + 5);
  detailY.style('width', '80px');
  describe(
    'a rotating white cylinder with limited Y
detail, with a slider that adjusts detailY'
  );
}

function draw() {
  background(205, 105, 94);
  rotateY(millis() / 1000);
  cylinder(20, 75, 16, detailY.value());
}
```

Функция cone(): рисует конус с заданным радиусом и высотой. DetailX и detailY определяют количество подразделений в x-измерении и y-измерении конуса. Больше подразделений заставляет конус казаться более гладким. Рекомендуемое максимальное значение для detailX равно 24. Использование значения больше 24 может вызвать предупреждение или замедлить работу браузера.

Синтаксис

```
cone([radius], [height], [detailX], [detailY], [cap]);
```

Параметры

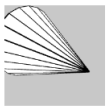
radius: радиус нижней поверхности (необязательно)

height: высота конуса (необязательно)

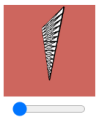
detailX: количество сегментов, чем больше сегментов, тем более гладкой геометрией по умолчанию является 24 (необязательно)

detailY: количество сегментов, чем больше сегментов, тем более гладкая геометрия по умолчанию равна 1 (необязательно)

Cap Boolean: рисовать ли основание конуса (Необязательно)

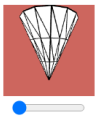


```
// draw a spinning cone
// with radius 40 and height 70
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a rotating white cone');
}

function draw() {
  background(200);
  rotateX(frameCount * 0.01);
  rotateZ(frameCount * 0.01);
  cone(40, 70);
}
```



edit reset

```
// slide to see how detailx works
let detailX;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailX = createSlider(3, 16, 3);
  detailX.position(10, height + 5);
  detailX.style('width', '80px');
  describe(
    'a rotating white cone with limited X detail,
with a slider that adjusts detailX'
  );
}

function draw() {
  background(205, 102, 94);
  rotateY(millis() / 1000);
  cone(30, 65, detailX.value(), 16);
}
```



edit reset

```
// slide to see how detailY works
let detailY;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailY = createSlider(3, 16, 3);
  detailY.position(10, height + 5);
  detailY.style('width', '80px');
  describe(
    'a rotating white cone with limited Y detail,
with a slider that adjusts detailY'
  );
}

function draw() {
  background(205, 102, 94);
  rotateY(millis() / 1000);
  cone(30, 65, 16, detailY.value());
}
```

Функция ellipsoid():рисует эллипсоид с заданным радиусом. DetailX и detailY определяют количество подразделений в x-измерении и y-измерении конуса. Больше делений делает эллипсоид более гладким. Избегайте номера детализации выше 150, это может привести к сбою браузера.

Синтаксис

```
ellipsoid([radiusx], [radiusy], [radiusz], [detailX], [detailY]);
```

Параметры

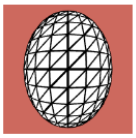
radiusx: x-радиус эллипсоида (необязательно)

radiusy: y-радиус эллипсоида (необязательно)

radiusz: z-радиус эллипсоида (необязательно)

detailX: количество сегментов, чем больше сегментов, тем сглаженная геометрия по умолчанию равна 24. Избегайте номера детализации выше 150, это может привести к сбою браузера. (Необязательный)

detailY: количество сегментов, чем больше сегментов, тем сглаженная геометрия по умолчанию равна 16. Избегайте номера детализации выше 150, это может привести к сбою браузера. (Необязательный)



```
// draw an ellipsoid
// with radius 30, 40 and 40.
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a white 3d ellipsoid');
}

function draw() {
  background(205, 105, 94);
  ellipsoid(30, 40, 40);
}
```

```
// slide to see how detailX works
let detailX;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailX = createSlider(2, 24, 12);
  detailX.position(10, height + 5);
  detailX.style('width', '80px');
  describe(
    'a rotating white ellipsoid with limited X
detail, with a slider that adjusts detailX'
  );
}

function draw() {
  background(205, 105, 94);
  rotateY(millis() / 1000);
  ellipsoid(30, 40, 40, detailX.value(), 8);
}
```

```
// slide to see how detailY works
let detailY;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailY = createSlider(2, 24, 6);
  detailY.position(10, height + 5);
  detailY.style('width', '80px');
  describe(
    'a rotating white ellipsoid with limited Y
detail, with a slider that adjusts detailY'
  );
}

function draw() {
  background(205, 105, 9);
  rotateY(millis() / 1000);
  ellipsoid(30, 40, 40, 12, detailY.value());
}
```

Функция torus(): рисует кольцо с заданным радиусом и радиусом трубы

DetailX и detailY определяют количество подразделений в x-измерении и y-измерении тора. Больше подразделений заставляет тор казаться более гладким. Максимальные значения по умолчанию для detailX и detailY - 24 и 16

соответственно. Установка для них относительно небольших значений, таких как 4 и 6, позволяет создавать новые формы, отличные от тора.

Синтаксис

```
torus([radius], [tubeRadius], [detailX], [detailY]);
```

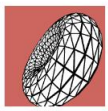
Параметры

radius: радиус всего кольца (необязательно)

tubeRadius: радиус трубки (необязательно)

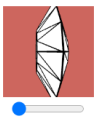
detailX: количество сегментов в x-измерении, чем больше сегментов, тем более гладкой геометрией по умолчанию является 24 (необязательно)

detailY: количество сегментов в y-измерении, чем больше сегментов, тем более гладкой геометрией по умолчанию является 16 (необязательно)



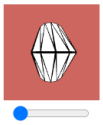
edit

```
// draw a spinning torus
// with ring radius 30 and tube radius 15
function setup() {
  createCanvas(100, 100, WEBGL);
  describe('a rotating white torus');
}

function draw() {
  background(205, 102, 94);
  rotateX(frameCount * 0.01);
  rotateY(frameCount * 0.01);
  torus(30, 15);
}
```



edit | reset | co

```
// slide to see how detailY works
let detailY;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailY = createSlider(3, 16, 3);
  detailY.position(10, height + 5);
  detailY.style('width', '80px');
  describe(
    'a rotating white torus with limited Y detail,
with a slider that adjusts detailY'
  );
}

function draw() {
  background(205, 102, 94);
  rotateY(millis() / 1000);
  torus(30, 15, 16, detailY.value());
}
```



edit | reset | cc

```
// slide to see how detailX works
let detailX;
function setup() {
  createCanvas(100, 100, WEBGL);
  detailX = createSlider(3, 24, 3);
  detailX.position(10, height + 5);
  detailX.style('width', '80px');
  describe(
    'a rotating white torus with limited X detail,
with a slider that adjusts detailX'
  );
}

function draw() {
  background(205, 102, 94);
  rotateY(millis() / 1000);
  torus(30, 15, detailX.value(), 12);
}
```

Простые геометрические преобразования

Функция rotate(): поворот фигуры на величину, указанную параметром угла. Эта функция учитывает angleMode, поэтому углы можно вводить в RADIANS или DEGREES.

Объекты всегда вращаются вокруг своей относительной позиции к началу координат, а положительные числа вращают объекты по часовой стрелке. Преобразования применяются ко всему, что происходит после, и последующие вызовы функции накапливают эффект. Например, вызов rotate (HALF_PI), а затем rotate (HALF_PI) аналогичен rotate (PI). Все преобразования сбрасываются, когда draw () начинается снова.

Технически rotate () умножает текущую матрицу преобразования на матрицу вращения. Эту функцию можно дополнительно контролировать с помощью push () и pop ().

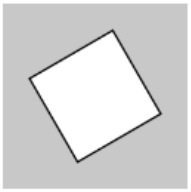
Синтаксис

```
rotate(angle, [axis]);
```

Параметры

angle: угол поворота, указанный в радианах или градусах, в зависимости от текущего angleMode

axis: (в 3d) задаём ось для вращения вокруг (необязательно)



```
translate(width / 2, height / 2);
rotate(PI / 3.0);
rect(-26, -26, 52, 52);
```

Функции rotateX(), rotateY(), rotateZ(): вращение вокруг оси X, Y и Z соответственно.

Синтаксис

```
rotateX(angle);
rotateY(angle);
rotateZ(angle);
```

Параметры

angle: угол поворота, указанный в радианах или градусах, в зависимости от текущего angleMode.

Функция scale(): увеличивает или уменьшает размер фигуры, расширяя и сужая вершины. Объекты всегда масштабируются от их относительного происхождения до системы координат. Значения шкалы указываются в десятичных процентах. Например, масштаб вызова функции (2.0) увеличивает размер фигуры на 200%.

Преобразования применяются ко всему, что происходит после и последующие вызовы функции умножают эффект. Например, вызов масштаба (2.0), а затем масштаба (1.5) такой же, как масштаб (3.0). Если scale () вызывается в draw (), преобразование сбрасывается, когда цикл начинается снова.

Использование этой функции с параметром z доступно только в режиме WEBGL. Эту функцию можно дополнительно контролировать с помощью push () и pop ().

Синтаксис

```
scale(s, [y], [z]);
scale(scales);
```

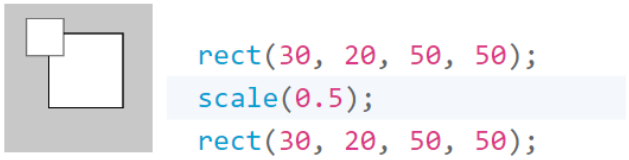
Параметры

s: процент для масштабирования объекта или процент для масштабирования объекта по оси X, если задано несколько аргументов

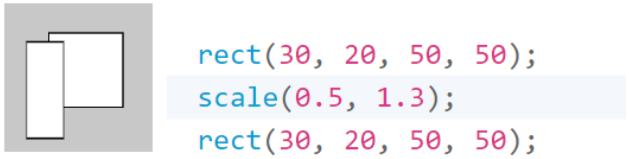
y: процент для масштабирования объекта по оси y (необязательно)

z: проценты для масштабирования объекта по оси z (только webgl) (необязательно)

scale: проценты по оси для масштабирования объекта



```
rect(30, 20, 50, 50);
scale(0.5);
rect(30, 20, 50, 50);
```



```
rect(30, 20, 50, 50);
scale(0.5, 1.3);
rect(30, 20, 50, 50);
```

Функция shearX(): обрезает форму вокруг оси X на величину, указанную параметром угла. Углы должны быть указаны в текущем angleMode. Объекты всегда срезаются вокруг их относительного положения относительно исходного положения, а положительные числа срезают объекты по часовой стрелке.

Преобразования применяются ко всему, что происходит после, и последующие вызовы функции накапливают эффект. Например, вызов shearX (PI / 2) и затем shearX (PI / 2) аналогичен shearX (PI). Если в draw () вызывается shearX (), преобразование сбрасывается, когда цикл начинается снова.

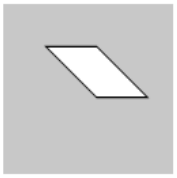
Технически shearX() умножает текущую матрицу преобразования на матрицу вращения. Эта функция может далее управляться функциями push () и pop ().

Синтаксис

```
shearX(angle);
```

Параметры

angle: угол сдвига, указанный в радианах или градусах, в зависимости от текущего angleMode.



```
translate(width / 4, height / 4);
shearX(PI / 4.0);
rect(0, 0, 30, 30);
```

Функция shearY(): обрезает форму вокруг оси Y на величину, указанную параметром угла. Углы должны быть указаны в текущем angleMode. Объекты всегда срезаются вокруг их относительного положения относительно исходного положения, а положительные числа срезают объекты по часовой стрелке.

Преобразования применяются ко всему, что происходит после, и последующие вызовы функции накапливают эффект. Например, вызов shearY (PI / 2) и затем shearY(PI / 2) аналогичен shearY (PI). Если в draw() вызывается shearY (), преобразование сбрасывается, когда цикл начинается снова.

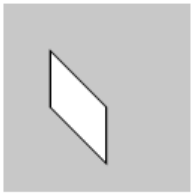
Технически shearY () умножает текущую матрицу преобразования на матрицу вращения. Эта функция может далее управляться функциями push() и pop().

Синтаксис

```
shearY(angle);
```

Параметры

angle: угол сдвига, указанный в радианах или градусах, в зависимости от текущего angleMode.



```
translate(width / 4, height / 4);
shearY(PI / 4.0);
rect(0, 0, 30, 30);
```

Функция translate(): определяет смещение объектов в окне отображения. Параметр x указывает движение влево/вправо, параметр y - движение вверх/вниз.

Преобразования являются кумулятивными и применяются ко всему, что происходит после, и последующие вызовы функции накапливают эффект. Например, вызов translate (50, 0) и затем translate (20, 0) аналогичен translate (70, 0). Если translate () вызывается в draw (), преобразование сбрасывается, когда цикл начинается снова. Эту функцию можно дополнительно контролировать с помощью push () и pop ().

Синтаксис

```
translate(x, y, [z]);
translate(vector);
```

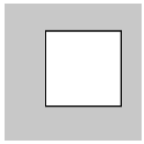
Параметры

x: левый / правый перевод

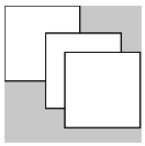
y: перевод вверх / вниз

z: прямой / обратный перевод (только webgl) (необязательно)

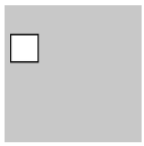
vector: задаёт вектор сдвига



```
translate(30, 20);
rect(0, 0, 55, 55);
```



```
rect(0, 0, 55, 55); // Draw rect at original 0,0
translate(30, 20);
rect(0, 0, 55, 55); // Draw rect at new 0,0
translate(14, 14);
rect(0, 0, 55, 55); // Draw rect at new 0,0
```



```
function draw() {
  background(200);
  rectMode(CENTER);
  translate(width / 2, height / 2);
  translate(p5.Vector.fromAngle(millis() / 1000,
40));
  rect(0, 0, 20, 20);
}
```

Функция applyMatrix(): Умножает текущую матрицу на ту, которая указана в параметрах. Это мощная операция, которая может выполнять эквивалент сдвига, масштабирования, сдвига и поворота одновременно. Вы можете узнать больше о матрицах преобразования в конспекте по графике.

Именование аргументов здесь следует за наименованием спецификации WHATWG и соответствует матрице преобразования вида:

Матрица преобразования, используемая при вызове applyMatrix

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

Синтаксис

```
applyMatrix(a, b, c, d, e, f);
```

Параметры

a: числа, которые определяют умножаемую матрицу 2x3

b: числа, которые определяют умножаемую матрицу 2x3

c: числа, которые определяют умножаемую матрицу 2x3

d: числа, которые определяют умножаемую матрицу 2x3

e: числа, которые определяют умножаемую матрицу 2x3

f: числа, которые определяют умножаемую матрицу 2x3

Переменные

Переменная предназначена для хранения информации в памяти, поэтому чтобы его можно было использовать позже в программе. А переменная можно использоваться несколько раз в одной программе, а значение может меняется во время работы программы.

Что-бы создать переменные используем следующий синтаксис:

var x; // Определяем переменную x

x = 12; // Присваеваем значение переменной x

var - ключевое слово которое показывает компилятору что следует определение переменной.

Переменные могут быть локальные и глобальные. Область действия переменной определяется просто: переменная создается внутри блока (код заключен в фигурные скобки: { и }) существует только внутри этого блока – такие переменные называются локальными.

Если переменная определена вне блоков (вне блока функции setup draw) то такая переменная называется глобальной.

В p5js существую специальные переменные называемые системными:

width – переменная, в которой хранится ширина холста рисования

height – переменная, в которой хранится высота холста рисования

movedX – переменная содержит горизонтальное движение мыши с момента последнего кадра

movedY – переменная содержит вертикальное движение мыши с момента последнего кадра

mouseX – переменная всегда содержит текущую горизонтальную позицию мыши относительно точки (0, 0) холста

mouseY – переменная всегда содержит текущую вертикальную позицию мыши относительно точки (0, 0) холста

pmouseX – системная переменная всегда содержит горизонтальное положение мыши или пальца в кадре, предшествующем текущему кадру, относительно точки (0, 0) холста

pmouseY – системная переменная всегда содержит вертикальное положение мыши или пальца в кадре, предшествующем текущему кадру, относительно точки (0, 0) холста

mouseIsPressed – системная переменная имеет значение true, если мышь нажата, и false, если нет

keyIsPressed системная переменная имеет значение true, если клавиша нажата, и false, если нет