

Графическая библиотека p5.js. Введение

Что такое p5.js?

P5.js – библиотека, написанная на языке программирования JavaScript, используемая для создания и просмотра интерактивных изображений с помощью простых графических примитивов. P5.js позволяет создавать графику на компьютере с использованием языка программирования. P5.js обеспечивает простую интеграцию кодов, написанных для веб-страниц, путем добавления кода, написанного в HTML-документе.

P5.js является бесплатным, открытым и независимым от платформы, поэтому приложение могут быть запущены на любой операционной системе, также p5.js имеет большое семейство связанных языков программирования и сред, которые показаны на рисунке 1.

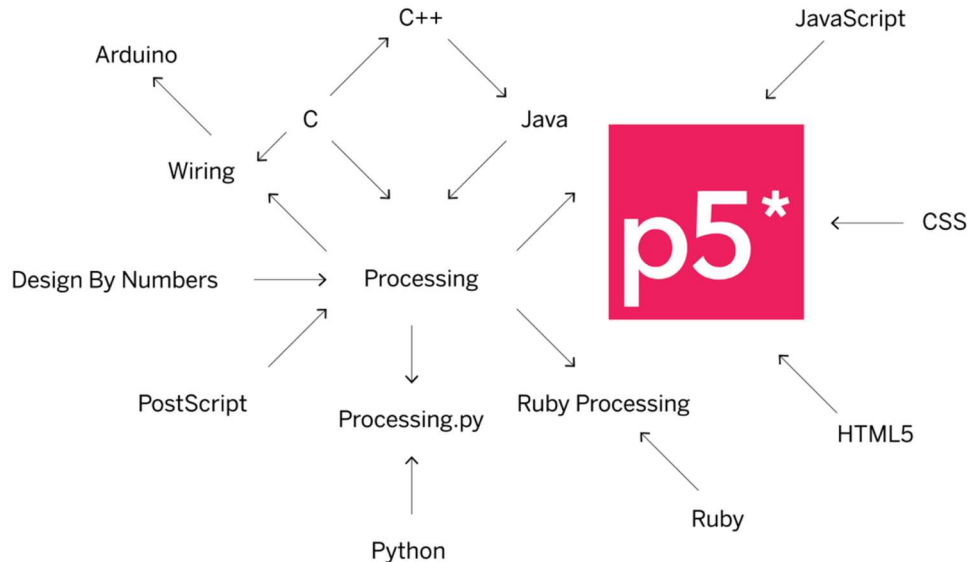


Рис. 1.1. Языки программирования и среды, связанные с p5.js

Создание простой программы в P5.js

Прежде чем начинать создавать собственные программы, мы должны знать, что любая фигура, созданная в среде p5.js связана с системой координат, началом системы координат в любой программе является верхний левый угол экрана. Вертикальная ось является осью Y, а горизонтальная ось – осью X. Увеличение значений координат X и Y показано на рисунке 2.

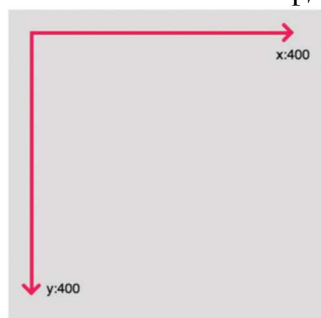


Рис. 1.2. Система координат в среде p5.js

Для создания простой программы в p5.js используется следующий шаблон:

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

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

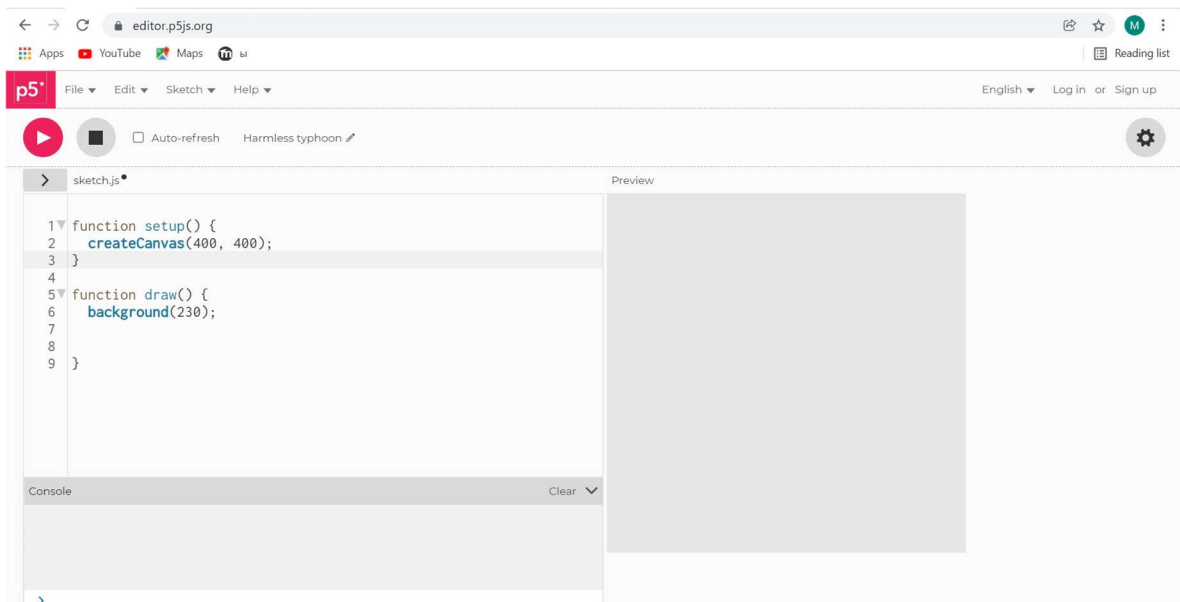


Рис. 1.3. Рабочее окно в среде p5.js

Функция **setup()** – вызывается только один раз в начале программы. Она используется для настройки начальных свойств рабочей среды, таких как размер и цвет экрана, а также для загрузки мультимедийных файлов при запуске программы, таких как изображения и шрифты. В каждой программе может быть только одна функция **setup()**, и она не должна вызываться после первоначального выполнения.

Примечание: Переменные, объявленные в **setup()**, недоступны в других функциях, включая **draw()**.

Пример:

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

createCanvas – создаёт элемент **canvas** (полотно) и задайте его размер в пикселях. Также вызывается только один раз в начале программы. Использование **createCanvas** несколько раз в коде приводит к непредсказуемому поведению. Если требуется более одного холста рисования, можно использовать функцию **createGraphics**.

Синтаксис функции **createCanvas** выглядит следующим образом:

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

где: **w** – число: ширина полотна;

h – число: высота полотна;

константа **renderer**: **P2D** – если начало координатной системы находится в левом верхнем углу экрана или **WEBGL** – если начало координатной системы находится в центре холста, характерно для 3D-графики (опционально).

Если **createCanvas ()** не используется в программе, холсту будет присвоено значение по умолчанию **100x100**.

Функция **draw ()** вызывается сразу после **setup()**, непрерывно выполняет строки кода, которые включены в ее состав, до завершения программы или пока не будет вызван **noLoop()**.

Примечание: Если функция **noLoop()** вызывается в **setup()**, функция **draw()** будет выполнена только один раз.

Синтаксис функции **draw ()**:

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

Функция **background ()** задает цвет используемый как фон холста. По умолчанию фон прозрачен. Эта функция обычно используется в `draw()` для удаления окна отображения в начале каждого кадра, но ее также можно использовать внутри функции `setup()`, чтобы установить фон на первом кадре анимации или если фон необходимо установить только один раз.

Цвет указывается в RGB, HSB или HSL в зависимости от `colorMode`. (По умолчанию используется режим – RGB, поэтому каждое значение находится в диапазоне $0 \div 255$).

Синтаксис функции:

`background(color)`

`background(colorstring, [a])`

`background(gray, [a])`

`background(v1, v2, v3, [a])`

`background(values)`

`background(image, [a])`

`color`: любое значение, созданное с помощью функции `color()`.

`colorstring String`: строка (название цвета на английском языке), возможные форматы: целое число `rgb ()` или `rgba ()`, процент `rgb ()` или `rgba ()`, 3-значный шестнадцатеричный, 6-значный шестнадцатеричный;

`a (число)`: прозрачность фона по отношению к текущей цветовой гамме (по умолчанию 0-255) (опционально);

`gray (число)`: значение цвета в оттенках серого, между черным и белым;

`v1 (число)`: указывает значение красного цвета или оттенка (в зависимости от текущей цветовой гаммы);

`v2 (число)`: задает значение зеленого цвета (оттенка) или значение насыщенности (в зависимости от текущей цветовой гаммы);

`v3 (число)`: задает значение синего цвета (оттенка) или значение яркости (в зависимости от текущей цветовой гаммы);

`values (число)`: массив, содержащий красную, зеленую, синюю и альфа-составляющие цветов;

`image`: изображение, созданное с помощью `loadImage()` или `createImage()`, которое может быть применено в качестве фона.

Создание простых примитивов 2D

Простые графические примитивы – это геометрические фигуры, которые могут быть созданы с помощью функций в графической библиотеке `p5.js`. Простейшими графическими примитивами являются 2D графические примитивы, на рисунке 1.4 показано соответствие точек геометрических фигур и параметров, которые должны быть указаны в качестве аргументов функций в программном коде.

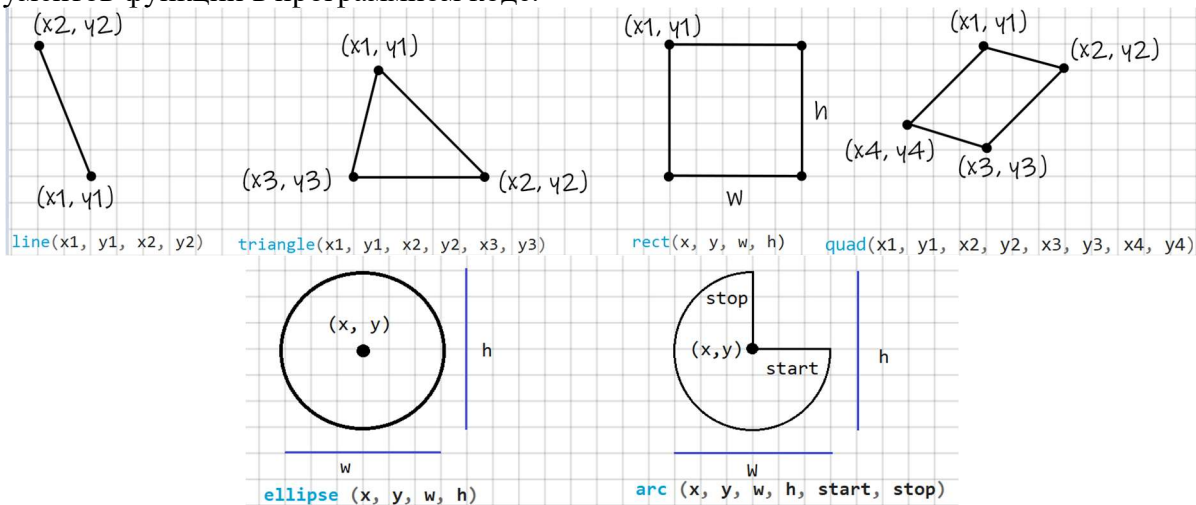
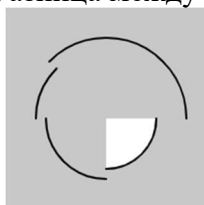
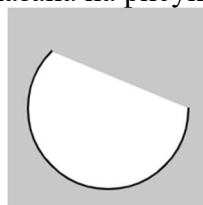


Рис. 1.4. Отношение между геометрическим точкам и параметрами функций `p5.js`

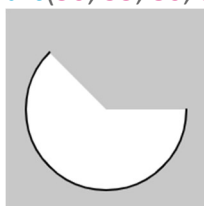
Функция **arc ()**: рисует дугу на экране. Если дуга рисуется, используя x , y , w , h , $start$ и $stop$, она будет нарисована и заполнена как дуга открытой окружности. Если указан параметр mod , дуга будет заполнена как открытый полукруг (OPEN), замкнутый полукруг (CHORD) или замкнутый круговой сегмент (PIE). То же самое можно применить и для функции `ellipseMode()`. Разница между этими режимами рисования показана на рисунке 1.5.



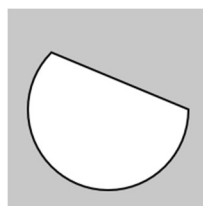
```
arc(50, 55, 50, 50, 0, HALF_PI);
noFill();
arc(50, 55, 60, 60, HALF_PI, PI);
arc(50, 55, 70, 70, PI, PI + QUARTER_PI);
arc(50, 55, 80, 80, PI + QUARTER_PI, TWO_PI);
```



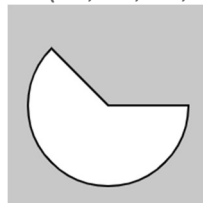
```
arc(50, 50, 80, 80, 0, PI + QUARTER_PI, OPEN);
```



```
arc(50, 50, 80, 80, 0, PI + QUARTER_PI);
```



```
arc(50, 50, 80, 80, 0, PI + QUARTER_PI, CHORD);
```



```
arc(50, 50, 80, 80, 0, PI + QUARTER_PI, PIE);
```

Рис. 1.5. Примеры рисования дуги (arc)

Дуга всегда рисуется по часовой стрелке, начальная точка ($start$), и конечная точка ($stop$) может использовать константы `r5.js` в соответствии со значениями, указанными в окружности, показанной на рисунке 1.6. Если начальная и конечная точки падают в одно и то же место, будет нарисован полный круг (эллипс). Обратите внимание, что ось Y увеличивается в направлении вниз, поэтому углы измеряются по часовой стрелке от положительного направления X .

Функция `arc ()` по умолчанию рисует в радианах, для перехода рисования в градусах используется `angleMode(DEGREES)`.

Синтаксис:

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

Параметры:

x (целое число): координата x центра дуги;

y (целое число): координата y центра дуги;

w (целое число): ширина окружности дуги;

h (целое число): высота окружности дуги;

$start$ (целое число): начальный угол дуги;

$stop$ (целое число): угол окончания дуги;

константа $mode$: параметр, определяющий способ рисования дуги. COORD, PIE или OPEN;

$detail$: опциональный параметр только для режима WebGL.

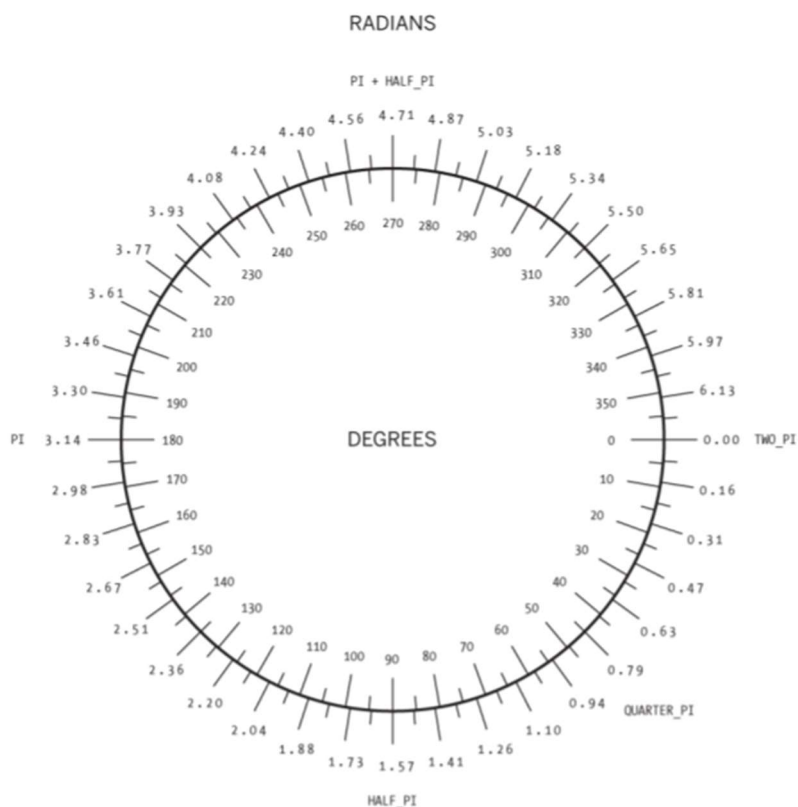


Рис. 1.6. Соотношение между константами, радианами и степенями в p5.js

Функция **ellipse()**: рисует эллипс (овал) на экране. Эллипс равной ширины и высоты представляет собой окружность. По умолчанию первые два параметра указывают координаты центра эллипса, а третий и четвертый параметры задают ширину и высоту. Если высота не указана, значение ширины используется как для ширины, так и для высоты. Если указана отрицательная высота или ширина, берется абсолютное значение. Источник можно изменить с помощью функции `ellipseMode()`.

Синтаксис:

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

`ellipse(x, y, w, h, detail)`

Параметры:

x (целое число): координата x центра эллипса;

y (целое число): координата y центра эллипса;

w (целое число): ширина эллипса;

h (целое число): высота эллипса;

detail: необязательный параметр, только 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 окружности. По умолчанию первые два параметра определяют координаты центра окружности, третий – диаметр окружности.

Синтаксис:

`circle(x, y, d)`

Параметры:

x (целое число): координата x центра круга;

y (целое число): координата y центра круга;

d (целое число): диаметр окружности.

На рисунке 1.7 приведен пример использования функций эллипса и окружности.

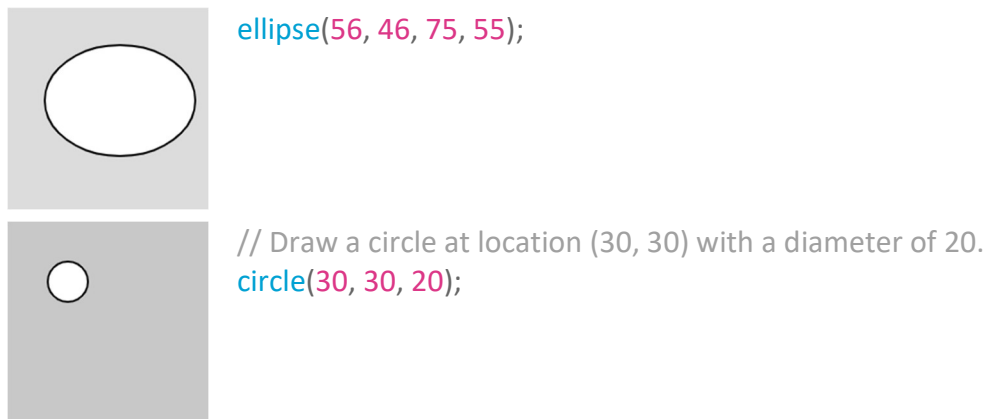


Рис. 1.7. Примеры программы и ее результат для функций эллипса (ellipse) и круга (circle)

Функция **line()**: рисует прямую линию между двумя точками на экране. Чтобы нарисовать линии разной толщины, можно использовать атрибут `stroke()`. Линия не может быть заполнена, поэтому атрибут `fill()` не может быть применен к ней. 2D-линии рисуются по умолчанию с шириной в один пиксель, если мы хотим указать толщину линии, мы используем функцию `strokeWeight()`.

Синтаксис:

`line(x1, y1, x2, y2)`

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

Параметры:

x1: координата x первой точки;

y1: координата y первой точки;

x2: координата x второй точки;

y2: координата y второй точки;

z1: координата z первой точки;

z2: координата z второй точки.

Пример:

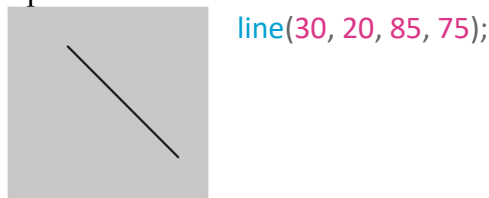


Рис. 1.8. Примеры программы для функции линия (line)

Функция **point()**: рисует точку, по координатам в 2D пространстве, размером в один пиксель. Первый параметр – это горизонтальное значение точки, второй – вертикальное значение точки. Цвет точки можно изменить с помощью функции `stroke()`. Размер точки изменяется с помощью функции `strokeWeight()`.

Синтаксис:

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

`point(coordinate_vector)`

Параметры:

x: координата x точки;

y: координата y точки;

z: координата z точки (опционально, только в режиме WebGL);

coordinate_vector: вектор координат.

Примеры программы и ее результат для функции point показаны на рисунке 1.9.

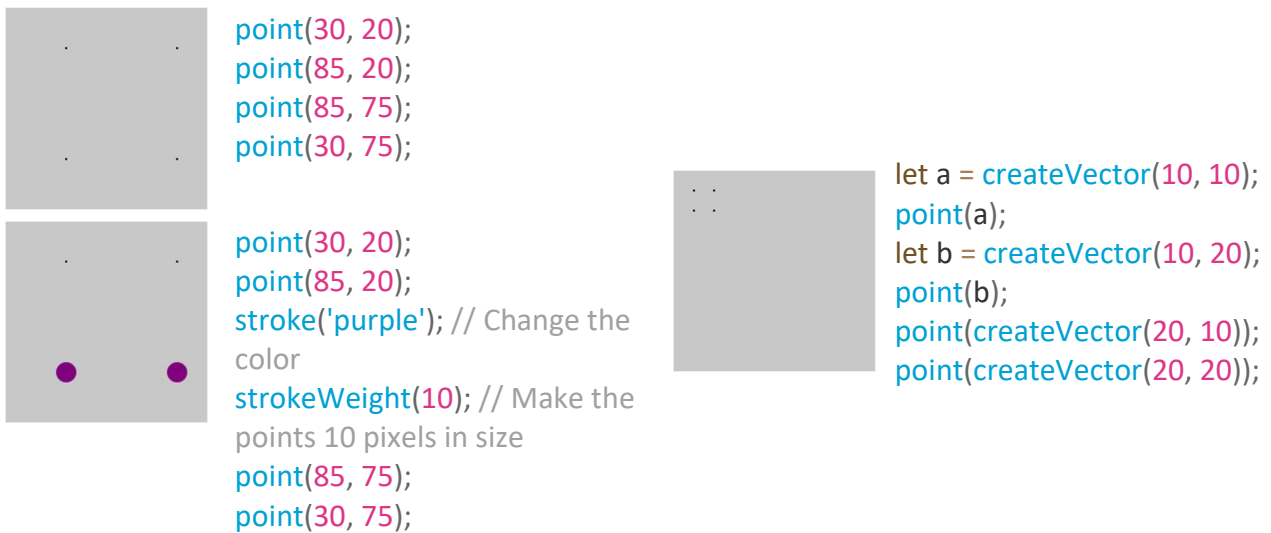


Рис. 1.9. Примеры программы для функции точка (point)

Функция **quad()**: рисует четырехугольник (четырёхсторонний многоугольник), углы между сторонами не ограничены девяносто градусами. Первая пара параметров (x1, y1) задает координаты первой точки, а последующие пары должны двигаться по часовой стрелке или против часовой стрелки вокруг predetermined формы. Аргументы z используются когда quad() работает 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: координата x первой точки;
- y1: координата y первой точки;
- x2: координата x второй точки;
- y2: координата y второй точки;
- x3: координата x третьей точки;
- y3: координата y третьей точки;
- x4: координата x четвертой точки;
- y4: координата y 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, четырехстороннюю фигуру с каждым углом в девяносто градусов. По умолчанию первые два параметра определяют положение верхнего левого угла, третий параметр указывает ширину, а четвертый параметр указывает высоту. Однако способ интерпретации этих параметров может быть изменен с помощью функции rectMode().

Пятый, шестой, седьмой и восьмой параметры, если они указаны, определяют радиус для верхнего левого, верхнего правого, нижнего правого и нижнего левого углов соответственно. Параметру радиуса угла присваивается значение радиуса, ранее указанное в списке параметров.

Синтаксис:

```

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

```

Параметры:

- x: координата x верхнего левого угла прямоугольника;

y: координата y верхнего левого угла прямоугольника;
 w: ширина прямоугольника;
 h: высота прямоугольника;
 tl: радиус левого верхнего угла (опционально);
 tr: радиус правого верхнего угла (опционально);
 br: радиус правого нижнего угла (опционально);
 bl: радиус левого нижнего угла (опционально);
 detailX (целое число): количество отрезков по оси X (для режима WebGL)
 (необязательно),
 detailY (целое число): количество отрезков по оси Y (для режима WebGL)
 (необязательно)ю

Примеры программы и их результаты показаны на рисунке 1.10.

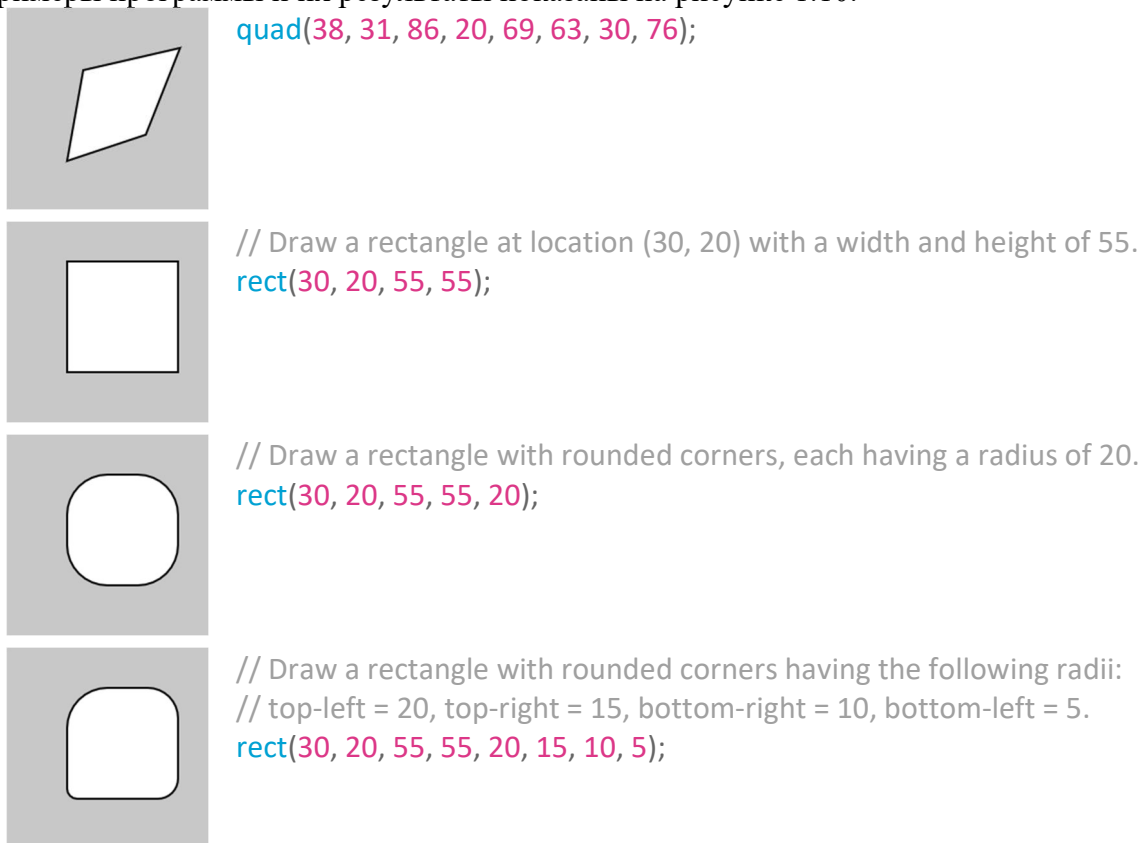


Рис. 1.10. Примеры программ для функций четырехугольник (quad) и прямоугольник (rect)

Функция **square()**: рисует квадрат на экране с каждым углом в девяносто градусов. Эта функция является частным случаем функции `rect()`, в которой ширина и высота одинаковы, и параметр `s` для размера ширины. По умолчанию первые два параметра координатируют верхний левый угол, третий – размер стороны квадрата. Однако способ интерпретации этих параметров может быть изменен с помощью функции `rectMode()`.

Четвертый, пятый, шестой и седьмой параметры, если они указаны, определяют радиус для верхнего левого, верхнего правого, нижнего правого и нижнего левого углов соответственно. Параметру радиуса угла присваивается значение радиуса, ранее указанное в списке параметров.

Синтаксис:

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

Параметры:

x: координата x верхнего левого угла квадрата;
 y: координата y верхнего левого угла квадрата;

s: размер стороны квадрата;
tl: радиус левого верхнего угла (опционально);
tr: радиус правого верхнего угла (опционально);
br: радиус правого нижнего угла (опционально);
bl: радиус левого нижнего угла (опционально).

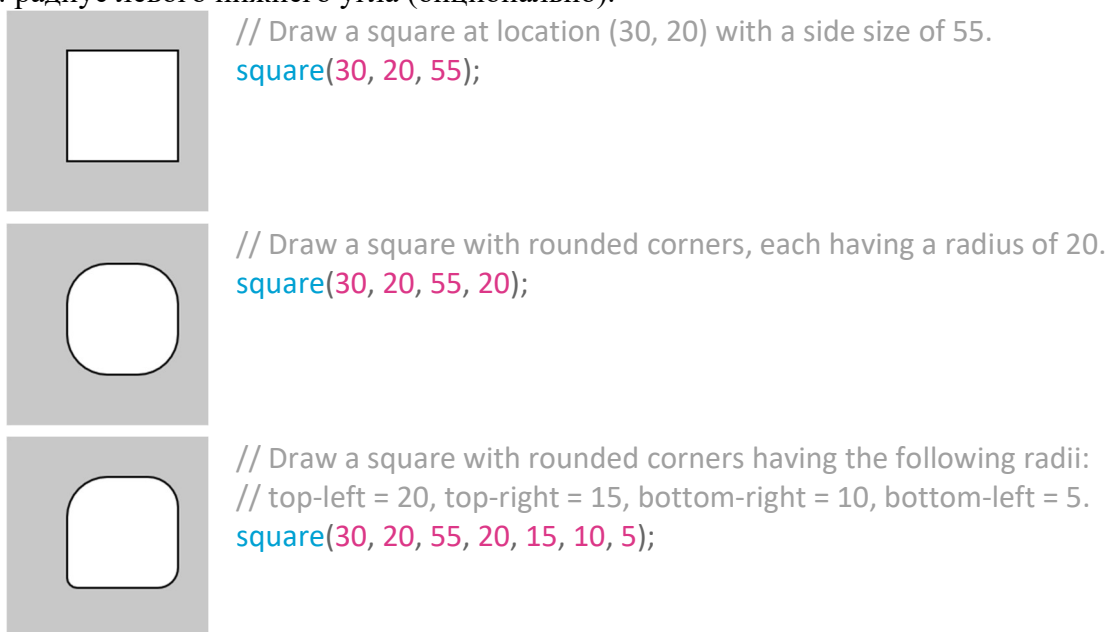


Рис. 1.11. Пример использования функции квадрат (square)

Функция **triangle()**: рисует треугольник. Аргумент функции указывают координаты первой точки, координаты второй точки, а последние два аргумента указывают координаты третьей точки.

Синтаксис:

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

Параметры:

x1: координата x первой точки;

y1: координата y первой точки;

x2: координата x второй точки;

y2: координата y второй точки;

x3: координата x третьей точки;

y3: координата y третьей точки.



Рис. 1.12. Пример использования функции треугольника (triangle)

Простые графические атрибуты

Особенности геометрических фигур, такие как цвет заливки фигуры, толщина и цвет линии обводки, способ соединения линий представляют собой простые графические атрибуты. Библиотека p5.js содержит список функций, позволяющих задавать или изменять эти атрибуты.

Основные графические атрибуты описаны ниже:

Функция **fill()**: Задаёт цвет, используемый для заливки фигур, все рисунки, нарисованные этой функцией, заполнены (окрашены) цветом, указанным в качестве параметра функции. Этот цвет указывается в плитках цвета RGB или HSB, в зависимости от текущего `colorMode()`. (По умолчанию используется цветовое пространство RGB, каждое значение варьируется от 0 до 255.)

Синтаксис:

`fill(v1, v2, v3, [alpha])`

`fill(value)`

`fill(gray, [alpha])`

`fill(values)`

`fill(color)`

Параметры:

v1 (целое число): значение красного цвета или оттенка (в зависимости от текущей цветовой гаммы);

v2 (целое число): значение зеленого цвета (оттенка) или значение насыщенности (в зависимости от текущей цветовой гаммы);

v3 (целое число): значение синего цвета (оттенка) или значение яркости (в зависимости от текущей цветовой гаммы);

alpha (целое число): (опционально);

value (строка): строка, указывающая цвет;

gray (целое число): значение оттенка серого;

values (числа []): таблица, содержащая цветовые компоненты: красный, зеленый, синий и альфа.

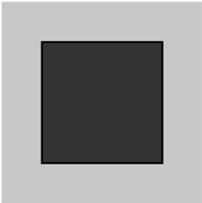
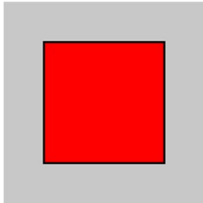
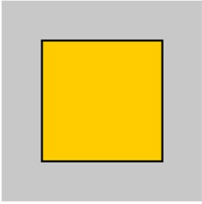
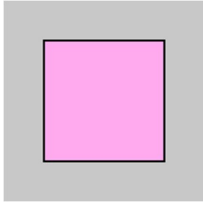
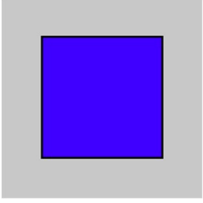

| | | | |
|---|--|---|---|
|  | <pre>// Grayscale integer value fill(51); rect(20, 20, 60, 60);</pre> |  | <pre>// Named SVG/CSS color string fill('red'); rect(20, 20, 60, 60);</pre> |
|  | <pre>// R, G & B integer values fill(255, 204, 0); rect(20, 20, 60, 60);</pre> |  | <pre>// three-digit hexadecimal RGB notation fill('#fae'); rect(20, 20, 60, 60);</pre> |
|  | <pre>// H, S & B integer values colorMode(HSB); fill(255, 204, 100); rect(20, 20, 60, 60);</pre> |  | <pre>// six-digit hexadecimal RGB notation fill('#222222'); rect(20, 20, 60, 60);</pre> |

Рис. 1.13. Примеры использования функции заливка (fill)

Если мы хотим, чтобы рисунок был прозрачным, мы используем функцию `noFill()`.

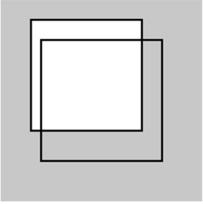
| | |
|---|---|
|  | <pre>rect(15, 10, 55, 55); noFill(); rect(20, 20, 60, 60)</pre> |
|---|---|

Рис. 1.14. Примеры использования функции без заливки (noFill)

Функция **stroke()**: задает цвет, используемый для рисования **обводки**, линий и краев рисунков. Этот цвет указывается в терминах цвета RGB или HSB, в зависимости от текущего `colorMode()` (по умолчанию цветовое пространство RGB, каждое значение варьируется от 0 до 255). Диапазон альфа-каналов по умолчанию также от 0 до 255.

Синтаксис:

`stroke(v1, v2, v3, [alpha])`

`stroke(value)`

`stroke(gray, [alpha])`

`stroke(values)`

`stroke(color)`

Параметры:

`v1` (целое число): значение красного цвета или оттенка (в зависимости от текущей цветовой гаммы);

`v2` (целое число): значение зеленого цвета (оттенка) или значение насыщенности (в зависимости от текущей цветовой гаммы);

`v3` (целое число): значение синего цвета (оттенка) или значение яркости (в зависимости от текущей цветовой гаммы);

`alpha` (целое число): (опционально);

`value` (строка): строка, указывающая цвет;

`gray` (целое число): значение оттенка серого;

`values` (числа []): таблица, содержащая цветовые компоненты: красный, зеленый, синий и альфа.

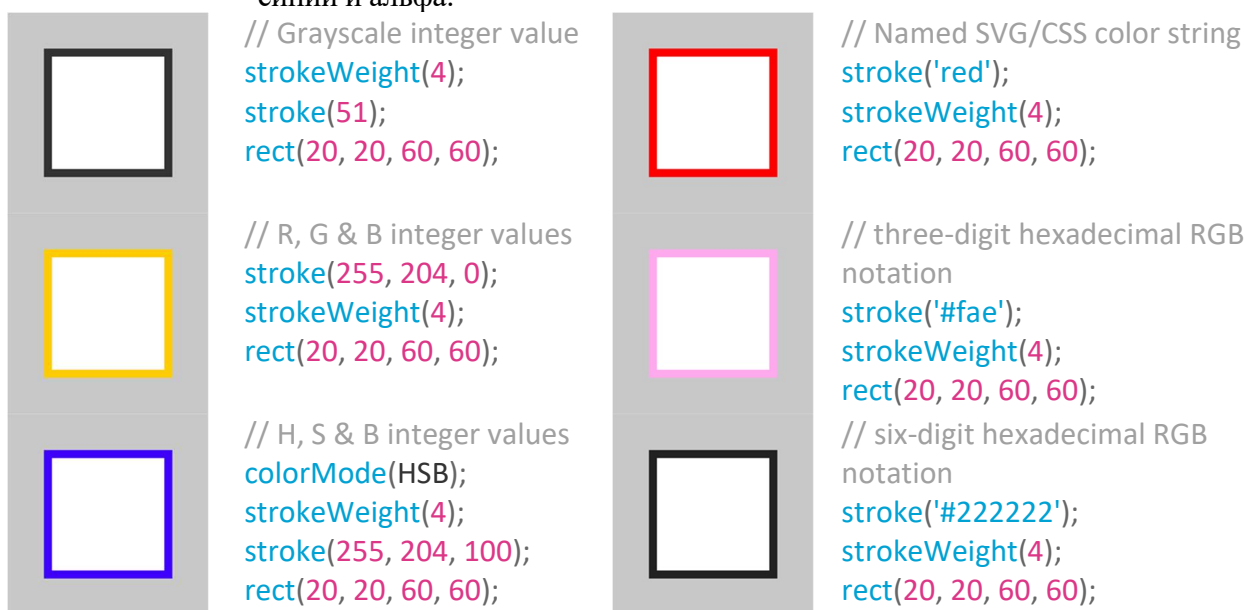


Рис. 1.15. Примеры использования штриха

Если мы хотим, чтобы рисунок был нарисован без обводки, используем функцию `noStroke()`.



Рис. 1.16. Примеры использования функции без обводки (`noStroke()`)

Помимо цвета, функция обводки может указывать следующее:

`strokeCap()`
`strokeJoin()`
`strokeWeight()`

Функция **strokeWeight(число)**: задает ширину линии. Все значения указываются в пикселях.

strokeCap(): задает стиль окончаний линий. Эти окончания могут быть: округлыми, квадратными, либо расширенными, каждая из которых задается соответствующими параметрами: ROUND, SQUARE и PROJECT. Значение по умолчанию – ROUND.

Примеры использования этих аргументов показаны на рисунке 1.17.

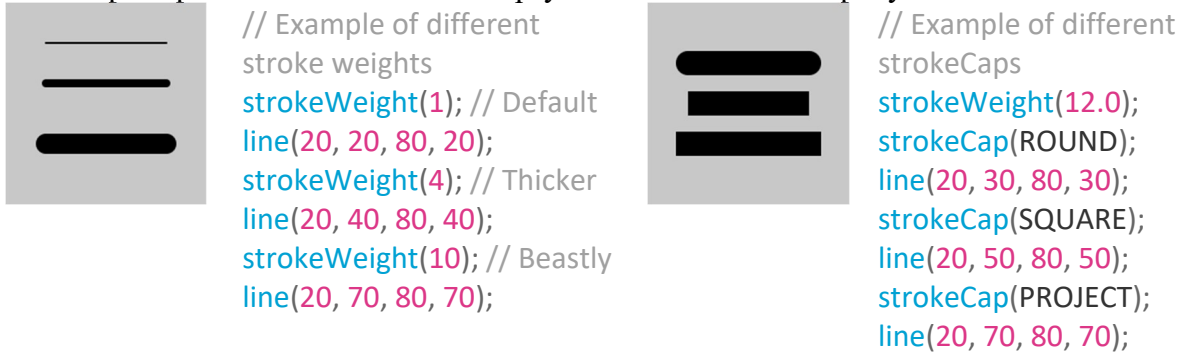


Рис. 1.17. Примеры утилизации ширины (`strokeWeight`) и окончаний (`strokeCap`) линий

Функция **strokeJoin(join)**: задает стиль соединения отрезков линии. Они могут быть указаны с соответствующими параметрами MITER, BEVEL и ROUND. По умолчанию он установлен как MITRE.

Разница между этими режимами показана на рисунке 1.11.



Рис. 1.18. Параметры функции соединения отрезков (`strokeJoin`)

При необходимости добавления текста может быть использована текстовая функция, которая может иметь разные параметры.

Функция **text()**: Рисует текст на экране, выводит информацию, указанную в первом параметре, на экране в положении, заданном дополнительными параметрами. Будет использоваться шрифт по умолчанию, если шрифт не установлен с помощью функции **textFont()**, то же самое с размером, будет использоваться размер по умолчанию, если размер шрифта не установлен через **textSize()**. Цвет текста можно изменить с помощью функции **fill()**. Изменение контура текста выполняется с помощью функций **stroke()** и **strokeWeight()**.

Текст можно выровнять с помощью функции **textAlign()**, которая предоставляет возможность делать выравнивание слева, справа и по центру.

Синтаксис:

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

Параметры:

str: текст который будет отображаться на экране;

x: координата x начальной точки текста;

y: координата y начальной точки текста;

x2 и y2: определяют прямоугольную область для отображения и может использоваться только со строковыми данными. Когда эти параметры указаны, они определяются на основе

текущего параметра `rectMode()`. Текст, который не полностью соответствует указанному прямоугольнику, не будет нарисован на экране. Если значения `x2` и `y2` не указаны, по умолчанию используется выравнивание базовой линии, что означает, что текст будет рисоваться вверх от `x` и `y`.

Основные атрибуты текста приведены на рисунке 1.19.




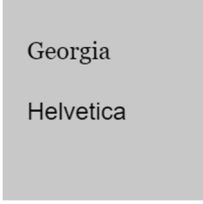
| | | | |
|---|--|--|--|
|  | <pre> textSize(32); text('word', 10, 30); fill(0, 102, 153); text('word', 10, 60); fill(0, 102, 153, 51); text('word', 10, 90); </pre> |  | <pre> textSize(12); text('Font Size 12', 10, 30); textSize(14); text('Font Size 14', 10, 60); textSize(16); text('Font Size 16', 10, 90); </pre> |
|  | <pre> strokeWeight(0); textSize(12); textStyle(NORMAL); text('Font Style Normal', 10, 15); textStyle(ITALIC); text('Font Style Italic', 10, 40); textStyle(BOLD); text('Font Style Bold', 10, 65); textStyle(BOLDITALIC); text('Font Style Bold Italic', 10, 90); </pre> |  | <pre> fill(0); textSize(12); textFont('Georgia'); text('Georgia', 12, 30); textFont('Helvetica'); text('Helvetica', 12, 60); </pre> |

Рис. 1.19. Примеры использования текстовых функций

Библиографические источники:

- <https://p5js.org/reference/>
- <https://github.com/processing/p5.js/wiki/p5.js-overview>
- «Make: Начало работы с p5.js» Лорен Маккарти, Кейси Риас, Бен Фрай;
- "Изучайте JavaScript с помощью p5.js" Энгин Арслан

Лабораторная работа 1

Тема: Изучение простых примитивов 2D-графики



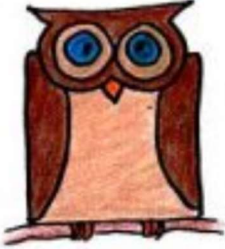
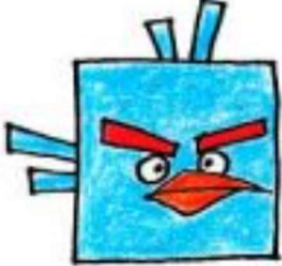



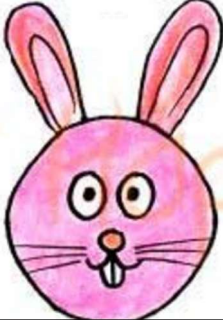
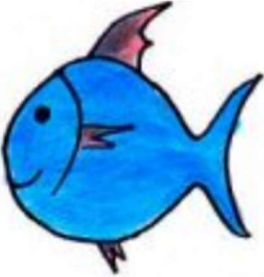
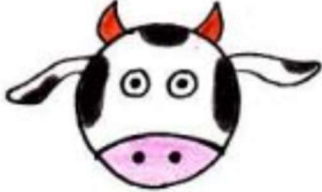


Цель работы: Получить практические знания в области синтеза статических 2D графических сцен, используя простые графические примитивы библиотеки `p5.js`.


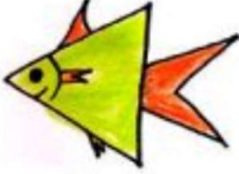
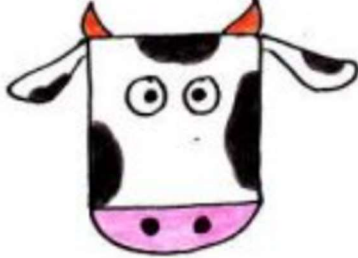



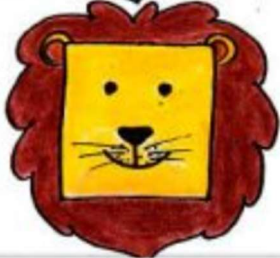

Задача работы:

1. Разработать программу для синтеза статической 2D-сцены с использованием не менее 6 графических примитивов разных типа `arc()`, `ellipse()`, `circle()`, `line()`, `point()`, `quad()`, `rect()`, `square()`, `triangle()`, примитивы должны быть с разными атрибутами, работа должна быть подписана (имя группы) в правом нижнем углу экрана.

2. Разработайте программу, которая создает персонажа в соответствии с вариантом, указанным преподавателем. Варианты приведены в таблице 1.1. Для создания этого изображения по шагам вы можете обратиться к странице <https://ja-rastu.ru/raskraski/uroki/1079-poshagovoe-risovanie-zhivotnyh-iz-geometriceskikh-figur.html>

Таблица 1.1. Варианты для выполнения лабораторной работы

| Вариант | Рисунок | Вариант | Рисунок |
|---------|---|---------|---|
| 1 |  | 11 |  |
| 2 |  | 12 |  |
| 3 |  | 13 |  |
| 4 |  | 14 |  |
| 5 |  | 15 |  |
| 6 |  | 16 |  |

| Вариант | Рисунок | Вариант | Рисунок |
|---------|--|---------|--|
| 7 |  | 17 |  |
| 8 |  | 18 |  |
| 9 |  | 19 |  |
| 10 |  | 20 |  |

Пример программы, выполненной в p5.js:

```

}
function draw() {
  background(220);
  CurrentY = Height - 20;

  //Realizam baza (1 parte)
  stroke(6, 21, 131);
  for (let i = 0; i < 20; i++) {
    line(0 + 20, CurrentY, Width - 20, CurrentY);
    CurrentY--;
  }

  // Realizam baza (2 parte)
  stroke(15, 23, 71);
  for (let i = 0; i < 20; i++) {
    line(0 + 20 + i, CurrentY, Width - 20 - i, CurrentY);
    CurrentY--;
  }

  //Основание двери
  CurrentY += 10;
  stroke(6, 21, 121);
  for (let i = 0; i < 550; i++) {
    line(0 + 40, CurrentY, Width - 40, CurrentY);
    CurrentY--;
  }

  //Колонки
  fill(6, 21, 121);
  stroke(15, 21, 71);

```

```

rect(40, CurrentY, 40, Height - 190);
rect(80, CurrentY, 3, Height - 190);
rect(83, CurrentY, 6, Height - 190);

rect(Width - 40, CurrentY, -40, Height - 190);
rect(Width - 80, CurrentY, -3, Height - 190);
rect(Width - 83, CurrentY, -6, Height - 190);

//Дверные ямы
for (let i = 0; i < 4; i++) {
  for (let j = 0; j < 2; j++) {
    stroke(129, 153, 193);
    line(110 + j * 100, Height - 80 - i * 130, 110 + j * 100, Height - 180 - i * 130);
    line(110 + j * 100, Height - 80 - i * 130, 190 + j * 100, Height - 80 - i * 130);
    stroke(10, 14, 45);
    line(110 + j * 100, Height - 180 - i * 130, 190 + j * 100, Height - 180 - i * 130);
    line(190 + j * 100, Height - 180 - i * 130, 190 + j * 100, Height - 80 - i * 130);
  }
}

//Средняя колонка
stroke(10, 14, 45);
rect(200, CurrentY, -3, Height - 190);
stroke(16, 31, 138);
rect(203, CurrentY, -3, Height - 190);
stroke(49, 65, 157);
rect(205, CurrentY, -1, Height - 190);

//Дверь
fill(240, 240, 240);
ellipse(210, 350, 8, 30);
fill(147, 127, 68);
circle(210, 410, 10);

stroke(10, 14, 45);
line(110, Height - 80 - 2 * 130, 110, Height - 180 - 2 * 130);
line(110, Height - 80 - 2 * 130, 190, Height - 80 - 2 * 130);
line(110, Height - 180 - 2 * 130, 190, Height - 180 - 2 * 130);
line(190, Height - 180 - 2 * 130, 190, Height - 80 - 2 * 130);
fill(240, 240, 240);
stroke(240, 240, 240);
rect(120, Height - 430, 60, 80);
fill(0, 0, 0);
noStroke();
textSize(5);
text('POLICE TELEPHONE', 125, Height - 420);
textSize(10);
text('FREE', 135, Height - 405);
textSize(5);
text('FOR USE OR', 132, Height - 395);
textSize(10);
text('PUBLIC', 130, Height - 380);
textSize(5);
text('ADVICE & ASSIS', 128, Height - 370);
textSize(7);
text('PULL TO OPEN', 125, Height - 355);

//Окна
for(let j = 0; j < 2; j++) {
  stroke(129, 153, 193);
  fill(240, 240, 240);
  rect(113 + j * 100, Height-565, 75,93);

  stroke(18, 34, 129);
  line(138 + j * 100, Height-565, 138 + j * 100,Height-473);
  line(163 + j * 100, Height-565, 163 + j * 100,Height-473);
  line(113 + j * 100, Height-519, 188 + j * 100,Height-519);
}

//Верхушка
CurrentY-=40
fill(6, 21, 121);
stroke(15, 21, 71);
rect(35, CurrentY, 330,50);
stroke(3, 11, 101);
strokeWeight(10);

```



```

fill(22, 27, 46);
rect(65, CurrentY, 270,50);
strokeWeight(1);

fill(255,255,255);
noStroke();
textSize(26);
text('POLICE', 90, CurrentY+35);
text('BOX',260, CurrentY+35);
textSize(12);
text('PUBLIC', 200, CurrentY+25);
text('CALL', 209, CurrentY+39);

CurrentY-=30
fill(6, 21, 121);
stroke(15, 21, 71);
rect(65, CurrentY, 270,30);

CurrentY-=20
rect(85, CurrentY, 230,20); }

```

Результат реализации программы:



Контрольные вопросы:

1. Назовите простые графические примитивы.
2. Как можно изменить атрибуты отображения графических примитивов?
3. Как можно написать текст графически?
4. Назовите стандартные форматы для изображений.