Opgave 1

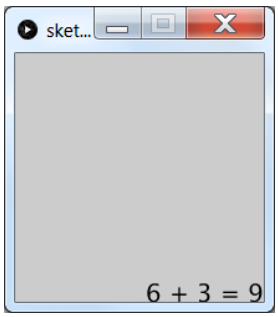
1A – 5 punten

Schrijf de programmaregel of programmaregels op die het getal 6 en 3 met elkaar vermenigvuldigd en de vermenigvuldiging met uitkomst in de message area van Processing afdrukt.

Hier moet dus letterlijk komen te staan: 6 * 3 = 18

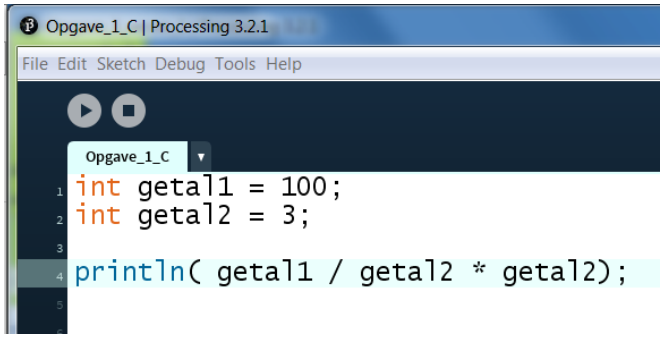
1B – 5 punten

Schrijf de programmaregels op die de volgende som 6 + 3 = *uitkomst* afdrukt rechtsonder in het grafisch window van Processing. Let op! Als de hoogte of breedte van het scherm (size) wijzigt moet de positie van de tekst hierop automatisch aangepast worden! Zo dus:



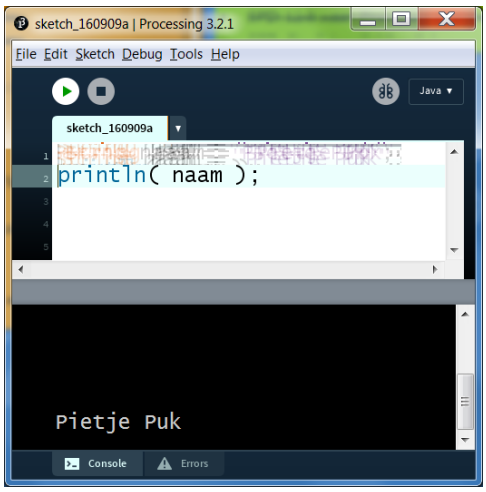
1C – 5 punten

Wat zal de uitput zijn in de text console van onderstaande code?



Opgave 2 – 10 punten

Wat moet er op de eerste regel staan, zodat er netjes *Pietje Puk* in de text console komt te staan?



Opgave 3 – 25 punten

Schrijf een programma dat in Processing de Griekse vlag weergeeft in een rechthoek. Een voorbeeld van de Griekse vlag staat hieronder:

Bron: http://www.crwflags.com/fotw/flags/gr.html

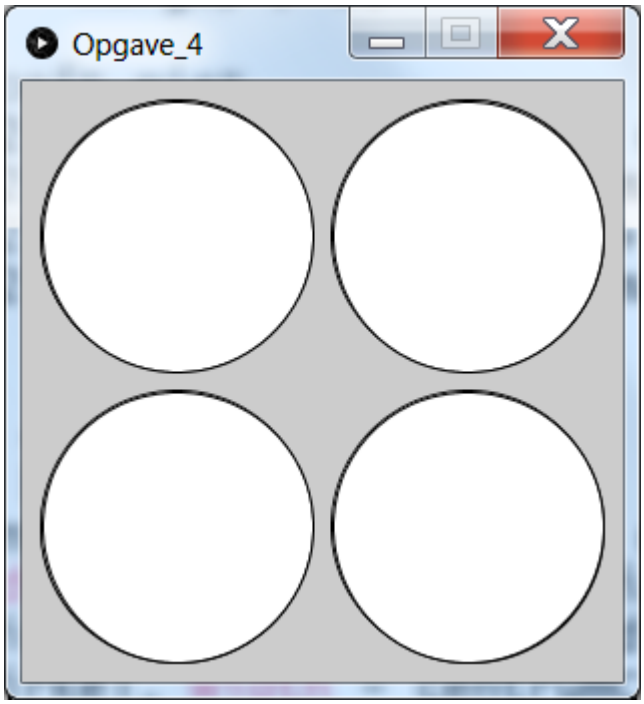
De verhouding hoogte breedte is 2 : 3. Verder, de Griekse vlag bestaat uit 9 horizontale strepen, afwisselend blauw en wit. Linksboven staat een blauw vierkant met een wit kruis.

Schrijf het programma zo dat de vlag bij iedere gegeven vlagbreedte goed getekend wordt. Je mag ervan uitgaan dat het grafische scherm groot genoeg is (size). Geef de volledige code van het programma.

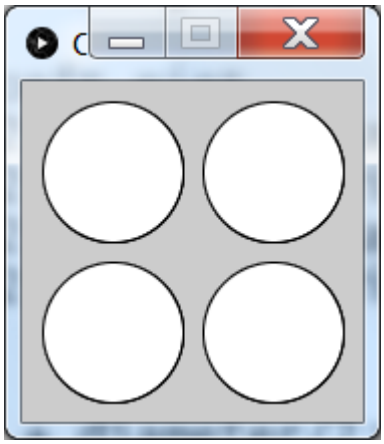
Opgave 4 – 30 punten

Schrijf een programma dat vier cirkels tekent in een vierkant scherm. De cirkels staan 10 pixels uit de rand en van elkaar af, m.a.w. er is steeds een ruimte van 10 pixels tussen de cirkels onderling en tussen de cirkels en de rand. De size is steeds hard gecodeerd, dus bijvoorbeeld size(300, 300) of size(500, 500).

Hieronder twee voorbeelden:



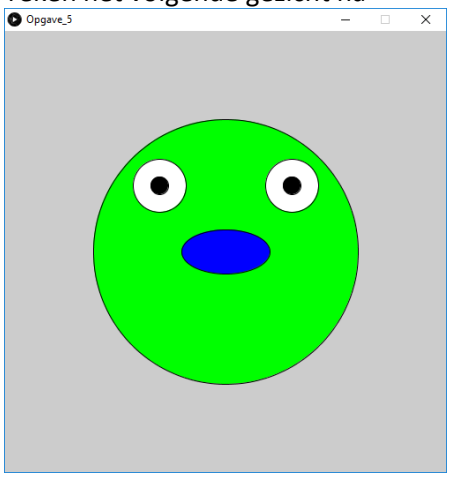
Scherm van 500 x 500 pixels



Scherm is 170 x 170 pixels

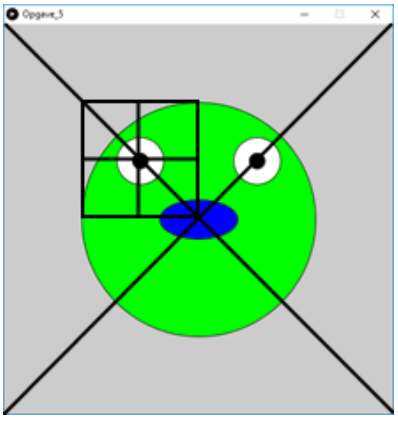
Opgave 5 – 20 punten

Teken het volgende gezicht na



Een aantal voorwaarden:

- De diameter van het gezicht is gegeven, bijv:
  int diameter = 300;
- De grootte van het scherm doet er niet toe, ga er maar vanuit dat het altijd groot genoeg is, bijv. size(500, 500);
- Het gezicht staat altijd in het midden van het scherm.
- De breedte van de neus is 1/3^{e} van de diameter van het gezicht
- De hoogte van de neus is de helft van de breedte van de neus
- De diameter van het hele oog is 1/5^{e} van de diameter van het gezicht
- De diameter van de iris (zwarte rondje) is 1/3^{e} van de diameter van het oog
- Het linkeroog is 1/4^{e} van de diameter van het gezicht naar links en 1/4^{e} van de diameter naar boven t.o.v. van het midden van het gezicht.
- Het rechteroog is 1/4^{e} van de diameter van het gezicht naar rechts en 1/4^{e} van de diameter naar boven t.o.v. van het midden van het gezicht.



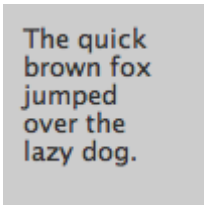
Bijlage: API-documentatie

Name text()

Examples


```
textSize(32);
text("word", 10, 30);
fill(0, 102, 153);
text("word", 10, 60);
fill(0, 102, 153, 51);
text("word", 10, 90);
```


```
size(100, 100, P3D);
textSize(32);
fill(0, 102, 153, 204);
text("word", 12, 45, -30);  // Specify a z-axis value
text("word", 12, 60);  // Default depth, no z-value specified
```




```
String s = "The quick brown fox jumped over the lazy dog.";
fill(50);
text(s, 10, 10, 70, 80);  // Text wraps within text box
```

Description Draws text to the screen. Displays the information specified in the first parameter on the screen in the position specified by the additional parameters. A default font will be used unless a font is set with the **textFont()** function and a default size will be used unless a font is set with **textSize()**. Change the color of the text with the **fill()** function. The text displays in relation to the **textAlign()** function, which gives the option to draw to the left, right, and center of the coordinates.

The **x2** and **y2** parameters define a rectangular area to display within and may only be used with string data. When these parameters are specified, they are interpreted based on the current **rectMode()** setting. Text that does not fit completely within the rectangle specified will not be drawn to the screen.

Note that Processing now lets you call **text()** without first specifying a PFont with **textFont()**. In that case, a generic sans-serif font will be used instead. (See the third example above.)

Syntax

```
text(c, x, y)
text(c, x, y, z)
text(str, x, y)
text(chars, start, stop, x, y)
text(str, x, y, z)
text(chars, start, stop, x, y, z)
text(str, x1, y1, x2, y2)
text(num, x, y)
text(num, x, y, z)
```

Parameters

c	char: the alphanumeric character to be displayed
x	float: x-coordinate of text
y	float: y-coordinate of text
z	float: z-coordinate of text
chars	char[]: the alphanumberic symbols to be displayed
start	int: array index at which to start writing characters
stop	int: array index at which to stop writing characters
x1	float: by default, the x-coordinate of text, see rectMode() for more info
y1	float: by default, the y-coordinate of text, see rectMode() for more info
x2	float: by default, the width of the text box, see rectMode() for more info
y2	float: by default, the height of the text box, see rectMode() for more info
num	int, or float: the numeric value to be displayed

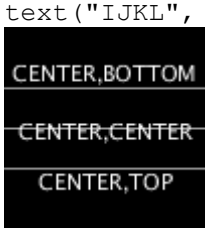
Returns void

Name textAlign()

Examples


```
background(0);
textSize(16);
textAlign(RIGHT);
text("ABCD", 50, 30);
textAlign(CENTER);
```

```
text("EFGH", 50, 50);
textAlign(LEFT);
text("IJKL", 50, 70);
```




```
background(0);
stroke(153);
textSize(11);
textAlign(CENTER, BOTTOM);
line(0, 30, width, 30);
text("CENTER,BOTTOM", 50, 30);
textAlign(CENTER, CENTER);
line(0, 50, width, 50);
text("CENTER,CENTER", 50, 50);
textAlign(CENTER, TOP);
line(0, 70, width, 70);
text("CENTER,TOP", 50, 70);
```

Description Sets the current alignment for drawing text. The parameters LEFT, CENTER, and RIGHT set the display characteristics of the letters in relation to the values for the **x** and **y** parameters of the **text()** function.

An optional second parameter can be used to vertically align the text. BASELINE is the default, and the vertical alignment will be reset to BASELINE if the second parameter is not used. The TOP and CENTER parameters are straightforward. The BOTTOM parameter offsets the line based on the current **textDescent()**. For multiple lines, the final line will be aligned to the bottom, with the previous lines appearing above it.

When using **text()** with width and height parameters, BASELINE is ignored, and treated as TOP. (Otherwise, text would by default draw outside the box, since BASELINE is the default setting. BASELINE is not a useful drawing mode for text drawn in a rectangle.)

The vertical alignment is based on the value of **textAscent()**, which many fonts do not specify correctly. It may be necessary to use a hack and offset by a few pixels by hand so that the offset looks correct. To do this as less of a hack, use some percentage of **textAscent()** or **textDescent()** so that the hack works even if you change the size of the font.

Syntax

```
textAlign(alignX)
textAlign(alignX, alignY)
```

Parameters

alignX int: horizontal alignment, either LEFT, CENTER, or RIGHT

alignY int: vertical alignment, either TOP, BOTTOM, CENTER, or BASELINE

Returns void

Name println()

Examples

```
String s = "The size is ";
int w = 1920;
int h = 1080;
println(s);
println(w, "x", h);

// This program writes to the console:
// The size is
// 1920 x 1080
```

```
print("begin- ");
float f = 0.3;
int i = 1024;
print("f is " + f + " and i is " + 1024);
String s = " -end";
println(s);

// This program writes to the console:
// "begin- f is 0.3 and i is 1024 -end"
```

Description The **println()** function writes to the console area, the black rectangle at the bottom of the Processing environment. This function is often helpful for looking at the data a program is producing. Each call to this function creates a new line of output. More than one parameter can be passed into the function by separating them with commas. Alternatively, individual elements can be separated with quotes ("") and joined with the addition operator (+).

Before Processing 2.1, **println()** was used to write array data to the console. Now, use **printArray()** to write array data to the console.

Note that the console is relatively slow. It works well for occasional messages, but does not support high-speed, real-time output (such as at 60 frames per second). It should also be noted, that a println() within a for loop can sometimes lock up the program, and cause the sketch to freeze.

Syntax

```
println()
println(what)
println(variables)
```

Parameters

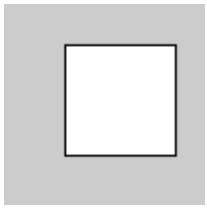
what Object, String, float, char, boolean, or byte: data to print to console

variables Object[]: list of data, separated by commas

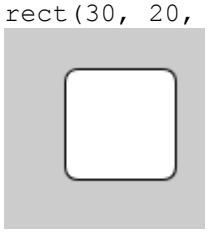
Returns void

Name rect()

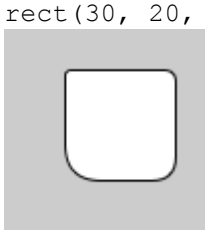
Examples



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



```
rect(30, 20, 55, 55, 7);
```



```
rect(30, 20, 55, 55, 3, 6, 12, 18);
```

Description Draws a rectangle to the screen. A rectangle is a four-sided shape with every angle at ninety degrees. By default, the first two parameters set the location of the upper-left corner, the third sets the width, and the fourth sets the height. The way these parameters are interpreted, however, may be changed with the **rectMode()** function.

To draw a rounded rectangle, add a fifth parameter, which is used as the radius value for all four corners.

To use a different radius value for each corner, include eight parameters. When using eight parameters, the latter four set the radius of the arc at each corner separately, starting with the top-left corner and moving clockwise around the rectangle.

Syntax

```
rect(a, b, c, d)
rect(a, b, c, d, r)
rect(a, b, c, d, tl, tr, br, bl)
```

Parameters

a float: x-coordinate of the rectangle by default

b float: y-coordinate of the rectangle by default

c float: width of the rectangle by default

d float: height of the rectangle by default

r float: radii for all four corners

tl float: radius for top-left corner

tr float: radius for top-right corner

br float: radius for bottom-right corner

bl float: radius for bottom-left corner

Returns void

Name constrain()

Examples

```
void draw()
{
  background(204);
  float mx = constrain(mouseX, 30, 70);
  rect(mx-10, 40, 20, 20);
}
```

Description Constrains a value to not exceed a maximum and minimum value.

Syntax `constrain(amt, low, high)`

Parameters

amt int, or float: the value to constrain

low int, or float: minimum limit

high int, or float: maximum limit

Returns float or int

Name noStroke()

Examples


```
noStroke();
rect(30, 20, 55, 55);
```

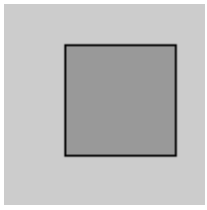
Description Disables drawing the stroke (outline). If both **noStroke()** and **noFill()** are called, nothing will be drawn to the screen.

Syntax `noStroke()`

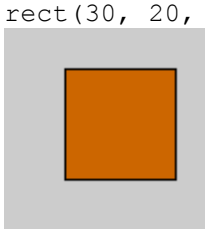
Returns void

Name fill()

Examples



```
fill(153);
rect(30, 20, 55, 55);
```



```
fill(204, 102, 0);
rect(30, 20, 55, 55);
```

Description Sets the color used to fill shapes. For example, if you run **fill(204, 102, 0)**, all subsequent shapes will be filled with orange. This color is either specified in terms of the RGB or HSB color depending on the current **colorMode()**. The default color space is RGB, with each value in the range from 0 to 255.

When using hexadecimal notation to specify a color, use "**#**" or "**0x**" before the values (e.g., **#CCFFAA** or **0xFFCCFFAA**). The **#** syntax uses six digits to specify a color (just as colors are typically specified in HTML and CSS). When using the hexadecimal notation starting with "**0x**", the hexadecimal value must be specified with eight characters; the first two characters define the alpha component, and the remainder define the red, green, and blue components.

The value for the "gray" parameter must be less than or equal to the current maximum value as specified by **colorMode()**. The default maximum value is 255.

To change the color of an image or a texture, use **tint()**.

Syntax

```
fill(rgb)
fill(rgb, alpha)
fill(gray)
fill(gray, alpha)
fill(v1, v2, v3)
fill(v1, v2, v3, alpha)
```

Parameters

rgb int: color variable or hex value

alpha float: opacity of the fill

gray float: number specifying value between white and black

v1 float: red or hue value (depending on current color mode)

v2 float: green or saturation value (depending on current color mode)

v3 float: blue or brightness value (depending on current color mode)

Returns	void