Reference

Filter by keywords...

Shortcuts

Data

Composite

`Array`

An array is a list of data

`ArrayList`

An `ArrayList` stores a variable number of objects

`FloatDict`

A simple table class to use a `String` as a lookup for a float value

`FloatList`

Helper class for a list of floats

`HashMap`

A `HashMap` stores a collection of objects, each referenced by a key

`IntDict`

A simple class to use a `String` as a lookup for an int value

`IntList`

Donate

`JSONArray`
A JSONArray is an ordered sequence of values

`JSONObject`
A `JSONObject` is an unordered collection of name/value pairs

`Object`
Objects are instances of classes

`String`
A string is a sequence of characters

`StringDict`
A simple class to use a `String` as a lookup for an `String` value

`StringList`
Helper class for a list of Strings

`Table`
Generic class for handling tabular data, typically from a CSV, TSV, or other sort of spreadsheet file

`TableRow`
Represents a single row of data values, stored in columns, from a `Table`

`XML`
This is the base class used for the Processing XML library, representing a single node of an `XML` tree

Array Functions

`append()`
Expands an array by one element and adds data to the new position

`arrayCopy()`
Copies an array (or part of an array) to another array

`concat()`
Concatenates two arrays

`expand()`
Increases the size of an array

`reverse()`
Reverses the order of an array

`shorten()`

Donate

sort()

Sorts an array of numbers from smallest to largest and puts an array of words in alphabetical order

splice()

Inserts a value or array of values into an existing array

subset()

Extracts an array of elements from an existing array

Conversion

binary()

Converts an `int`, `byte`, `char`, or `color` to a `String` containing the equivalent binary notation

boolean()

Converts an `int` or `String` to its boolean representation

byte()

Converts any value of a primitive data type (`boolean`, `byte`, `char`, `color`, `double`, `float`, `int`, or `long`) to its byte representation

char()

Converts any value of a primitive data type (`boolean`, `byte`, `char`, `color`, `double`, `float`, `int`, or `long`) to its numeric character representation

float()

Converts an `int` or `String` to its floating point representation

hex()

Converts a `byte`, `char`, `int`, or `color` to a `String` containing the equivalent hexadecimal notation

int()

Converts any value of a primitive data type (`boolean`, `byte`, `char`, `color`, `float`, `int`, or `long`) or String to its integer representation

str()

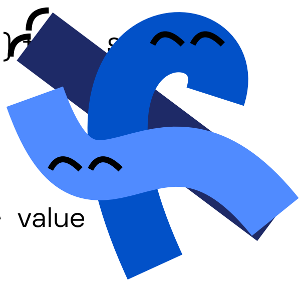
Converts a value of a primitive data type (`boolean`, `byte`, `char`, `int`, or `float`) representation

unbinary()

Converts a `String` representation of a binary number to its equivalent `integer` value

unhex()

Converts a `String` representation of a hexadecimal number to its equivalent integer va



Search

`boolean`

Datatype for the Boolean values `true` and `false`

`byte`

Datatype for bytes, 8 bits of information storing numerical values from 127 to -128

`char`

Datatype for characters, typographic symbols such as A, d, and $

`color`

Datatype for storing color values

`double`

Datatype for floating-point numbers larger than those that can be stored in a `float`

`float`

Data type for floating-point numbers, e

`int`

Datatype for integers, numbers without a decimal point

`long`

Datatype for large integers

String Functions

`join()`

Combines an array of `Strings` into one `String`, each separated by the character(s) used for the `separator` parameter

`matchAll()`

This function is used to apply a regular expression to a piece of text

`match()`

The function is used to apply a regular expression to a piece of text, and return matching groups (elements found inside parentheses) as a `String` array

`nf()`

Utility function for formatting numbers into strings

`nfc()`

Utility function for formatting numbers into strings and placing appropriate commas tc Donate its of 1000 `nfs()`

Utility function for formatting numbers into strings

`splitTokens()`

The `splitTokens()` function splits a `String` at one or many character ”tokens”

`split()`

The `split()` function breaks a string into pieces using a character or string as the divider

`trim()`

Removes whitespace characters from the beginning and end of a `String`

Input

Files

`BufferedReader`

A `BufferedReader` object is used to read files line-by-line as individual `String` objects

`createInput()`

This is a function for advanced programmers to open a Java `InputStream`

`createReader()`

Creates a `BufferedReader` object that can be used to read files line-by-line as individual `String` objects

`launch()`

Attempts to open an application or file using your platform’s launcher

`loadBytes()`

Reads the contents of a file or url and places it in a byte array

`loadJSONArray()`

Takes a `String`, parses its contents, and returns a `JSONArray`

`loadJSONObject()`

Loads a JSON from the data folder or a URL, and returns a `JSONObject`

`loadStrings()`

Reads the contents of a file or url and creates a `String` array of its individual lines

Donate

`loadTable()`

Reads the contents of a file or URL and creates an `XML` object with its values

`parseJSONArray()`
Takes a `String`, parses its contents, and returns a `JSONArray`

`parseJSONObject()`
Takes a `String`, parses its contents, and returns a `JSONObject`

`parseXML()`
Converts String content to an `XML` object

`selectFolder()`
Opens a platform-specific file chooser dialog to select a folder

`selectInput()`
Open a platform-specific file chooser dialog to select a file for input

Time & Date

`day()`
Returns the current day as a value from 1 - 31

`hour()`
Returns the current hour as a value from 0 - 23

`millis()`
Returns the number of milliseconds (thousandths of a second) since starting an applet

`minute()`
Returns the current minute as a value from 0 - 59

`month()`
Returns the current month as a value from 1 - 12

`second()`
Returns the current second as a value from 0 - 59

`year()`
Returns the current year as an integer (2003, 2004, 2005, etc)

Donate

Keyboard

`key`

`keyCode`

Used to detect special keys such as the UP, DOWN, LEFT, RIGHT arrow keys and ALT, CONTROL, SHIFT

`keyPressed`

The boolean system variable that is `true` if any key is pressed and `false` if no keys are pressed

`keyPressed()`

Called once every time a key is pressed

`keyReleased()`

Called once every time a key is released

`keyTyped()`

Called once every time a key is pressed, but action keys such as Ctrl, Shift, and Alt are ignored

Mouse

`mouseButton`

Shows which mouse button is pressed

`mouseClicked()`

Called once after a mouse button has been pressed and then released

`mouseDragged()`

Called once every time the mouse moves and a mouse button is pressed

`mouseMoved()`

Called every time the mouse moves and a mouse button is not pressed

`mousePressed`

Variable storing if a mouse button is pressed

`mousePressed()`

Called once after every time a mouse button is pressed

`mouseReleased()`

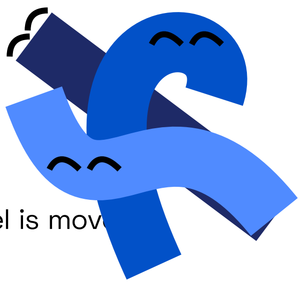
Called every time a mouse button is released

`mouseWheel()`

The code within the `mouseWheel()` event function is run when the mouse wheel is mov

`mouseX`

The system variable that always contains the current horizontal coordinate of the mous



`pmouseX`

The system variable that always contains the horizontal position of the mouse in the frame previous to the current frame

`pmouseY`

The system variable that always contains the vertical position of the mouse in the frame previous to the current frame

Constants

`HALF_PI`

HALF_PI is a mathematical constant with the value 1.57079632679489661923

`PI`

PI is a mathematical constant with the value 3.14159265358979323846

`QUARTER_PI`

QUARTER_PI is a mathematical constant with the value 0.7853982

`TAU`

An alias for `TWO_PI`

`TWO_PI`

TWO_PI is a mathematical constant with the value 6.28318530717958647693

Typography

`PFont`

Grayscale bitmap font class used by Processing

Loading & Displaying

`createFont()`

Dynamically converts a font to the format used by Processing

`loadFont()`

Loads a font into a variable of type `PFont`

Donate

`text()`
Draws text to the screen

Attributes

`textAlign()`
Sets the current alignment for drawing text

`textLeading()`
Sets the spacing between lines of text in units of pixels

`textMode()`
Sets the way text draws to the screen

`textSize()`
Sets the current font size

`textWidth()`
Calculates and returns the width of any character or text string

Metrics

`textAscent()`
Returns ascent of the current font at its current size

`textDescent()`
Returns descent of the current font at its current size

Rendering

`PGraphics`
Main graphics and rendering context, as well as the base API implementation for p

`blendMode()`
Blends the pixels in the display window according to a defined mode

`clip()`
Limits the rendering to the boundaries of a rectangle defined by the parameters

`createGraphics()`

`hint()`

This function is used to enable or disable special features that control how graphics are drawn

`noClip()`

Disables the clipping previously started by the `clip()` function

Shaders

`PShader`

This class encapsulates a GLSL shader program, including a vertex and a fragment shader

`loadShader()`

Loads a shader into the `PShader` object

`resetShader()`

Restores the default shaders

`shader()`

Applies the shader specified by the parameters

Image

`PImage`

Datatype for storing images

`createImage()`

Creates a new `PImage` (the datatype for storing images)

Pixels

`blend()`

Copies a pixel or rectangle of pixels using different blending modes

Donate

`copy()`

Copies the entire image

`filter()`

Converts the image to grayscale or black and white

`get()`

loadPixels()

Loads the pixel data for the display window into the `pixels[]` array

mask()

Masks part of an image with another image as an alpha channel

pixels[]

Array containing the values for all the pixels in the display window

set()

Writes a color to any pixel or writes an image into another

updatePixels()

Updates the display window with the data in the `pixels[]` array

Loading & Displaying

imageMode()

Modifies the location from which images draw

image()

Displays images to the screen

loadImage()

Loads an image into a variable of type `PImage`

noTint()

Removes the current fill value for displaying images and reverts to displaying images with their original hues

requestImage()

Loads images on a separate thread so that your sketch does not freeze while images load during `setup()`

tint()

Sets the fill value for displaying images

Textures

textureMode()

Sets the coordinate space for texture mapping

textureWrap()

Donate

Sets a texture to be applied to vertex points

Shape

`PShape`

Datatype for storing shapes

`createShape()`

The `createShape()` function is used to define a new shape

`loadShape()`

Loads geometry into a variable of type `PShape`

2d Primitives

`arc()`

Draws an arc in the display window

`circle()`

Draws a circle to the screen

`ellipse()`

Draws an ellipse (oval) in the display window

`line()`

Draws a line (a direct path between two points) to the screen

`point()`

Draws a point, a coordinate in space at the dimension of one pixel

`quad()`

A quad is a quadrilateral, a four sided polygon

`rect()`

Draws a rectangle to the screen

`square()`

Draws a square to the screen

`triangle()`

A triangle is a plane created by connecting three points

Donate

Vertex

`beginContour()`

Begins recording vertices for the shape

`beginShape()`

Using the `beginShape()` and `endShape()` functions allow creating more complex forms

`bezierVertex()`

Specifies vertex coordinates for Bezier curves

`curveVertex()`

Specifies vertex coordinates for curves

`endContour()`

Stops recording vertices for the shape

`endShape()`

the companion to `beginShape()` and may only be called after `beginShape()`

`quadraticVertex()`

Specifies vertex coordinates for quadratic Bezier curves

`vertex()`

All shapes are constructed by connecting a series of vertices

Curves

`bezierDetail()`

Sets the resolution at which Beziers display

`bezierPoint()`

Evaluates the Bezier at point t for points a, b, c, d

`bezierTangent()`

Calculates the tangent of a point on a Bezier curve

`bezier()`

Draws a Bezier curve on the screen

`curveDetail()`

Sets the resolution at which curves display

`curvePoint()`

Evaluates the curve at point t for points a, b, c, d

Donate

`curveTightness()`

Modifies the quality of forms created with `curve()` and `curveVertex()`

`curve()`

Draws a curved line on the screen

3D Primitives

`box()`

A box is an extruded `rectangle`

`sphereDetail()`

Controls the detail used to render a sphere by adjusting the number of vertices of the sphere mesh

`sphere()`

A sphere is a hollow ball made from tessellated triangles

Attributes

`ellipseMode()`

The origin of the ellipse is modified by the `ellipseMode()` function

`rectMode()`

Modifies the location from which rectangles draw

`strokeCap()`

Sets the style for rendering line endings

`strokeJoin()`

Sets the style of the joints which connect line segments

`strokeWeight()`

Sets the width of the stroke used for lines, points, and the border around shapes

Loading & Displaying

`shapeMode()`

Modifies the location from which shapes draw

`shape()`

Displays shapes to the screen

Math

`PVector`

A class to describe a two or three dimensional vector

Calculation

`abs()`

Calculates the absolute value (magnitude) of a number

`ceil()`

Calculates the closest int value that is greater than or equal to the value of the parameter

`constrain()`

Constrains a value to not exceed a maximum and minimum value

`dist()`

Calculates the distance between two points

`exp()`

Returns Euler's number *e* (2.71828...) raised to the power of the `value` parameter

`floor()`

Calculates the closest int value that is less than or equal to the value of the parameter

`lerp()`

Calculates a number between two numbers at a specific increment

`log()`

Calculates the natural logarithm (the base-*e* logarithm) of a number

`mag()`

Calculates the magnitude (or length) of a vector

`map()`

Re-maps a number from one range to another

`max()`

Determines the largest value in a sequence of numbers

`min()`

Determines the smallest value in a sequence of numbers

Donate

`pow()`

Facilitates exponential expressions

`round()`

Calculates the integer closest to the `value` parameter

`sq()`

Squares a number (multiplies a number by itself)

`sqrt()`

Calculates the square root of a number

Trigonometry

`acos()`

The inverse of `cos()`, returns the arc cosine of a value

`asin()`

The inverse of `sin()`, returns the arc sine of a value

`atan2()`

Calculates the angle (in radians) from a specified point to the coordinate origin as measured from the positive x-axis

`atan()`

The inverse of `tan()`, returns the arc tangent of a value

`cos()`

Calculates the cosine of an angle

`degrees()`

Converts a radian measurement to its corresponding value in degrees

`radians()`

Converts a degree measurement to its corresponding value in radians

`sin()`

Calculates the sine of an angle

`tan()`

Calculates the ratio of the sine and cosine of an angle

Donate

Operators

Combines addition with assignment

`+ (addition)`

Adds two values or concatenates string values

`-- (decrement)`

Substracts the value of an integer variable by 1

`/ (divide)`

Divides the value of the first parameter by the value of the second parameter

`/= (divide assign)`

Combines division with assignment

`++ (increment)`

Increases the value of an integer variable by 1

`- (minus)`

Subtracts one value from another and may also be used to negate a value

`% (modulo)`

Calculates the remainder when one number is divided by another

`* (multiply)`

Multiplies the values of the two parameters

`*= (multiply assign)`

Combines multiplication with assignment

`-= (subtract assign)`

Combines subtraction with assignment

Bitwise Operators

`& (bitwise AND)`

Compares each corresponding bit in the binary representation of the values

`| (bitwise OR)`

Compares each corresponding bit in the binary representation of the values

`<< (left shift)`

Shifts bits to the left

`>> (right shift)`

Shifts bits to the right

Donate

Random

`noiseDetail()`

Adjusts the character and level of detail produced by the Perlin noise function

`noiseSeed()`

Sets the seed value for `noise()`

`noise()`

Returns the Perlin noise value at specified coordinates

`randomGaussian()`

Returns a float from a random series of numbers having a mean of 0 and standard deviation of 1

`randomSeed()`

Sets the seed value for `random()`

`random()`

Generates random numbers

Output

Files

`PrintWriter`

Allows characters to print to a text-output stream

`beginRaw()`

To create vectors from 3D data, use the `beginRaw()` and `endRaw()` commands

`beginRecord()`

Opens a new file and all subsequent drawing functions are echoed to this file as well as the display window

`createOutput()`

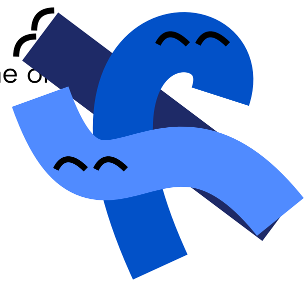
Similar to `createInput()`, this creates a Java `OutputStream` for a given filename o

`createWriter()`

Creates a new file in the sketch folder, and a `PrintWriter` object to write to it

`endRaw()`

Complement to `beginRaw()`; they must always be used together



`endRecord()`

`saveBytes()`

Opposite of `loadBytes()`, will write an entire array of bytes to a file

`saveJSONArray()`

Writes the contents of a `JSONArray` object to a file

`saveJSONObject()`

Writes the contents of a `JSONObject` object to a file

`saveStream()`

Save the contents of a stream to a file in the sketch folder

`saveStrings()`

Writes an array of strings to a file, one line per string

`saveTable()`

Writes the contents of a `Table` object to a file

`saveXML()`

Writes the contents of an `XML` object to a file

`selectOutput()`

Opens a platform-specific file chooser dialog to select a file for output

Text Area

`printArray()`

Writes array data to the text area of the Processing environment's console.

`print()`

Writes to the console area of the Processing environment

`println()`

Writes to the text area of the Processing environment's console

Image

`saveFrame()`

Saves a numbered sequence of images, one image each time the function is run

`save()`

Saves an image from the display window

Donate

Color

Creating & Reading

`alpha()`
Extracts the alpha value from a color

`blue()`
Extracts the blue value from a color, scaled to match current `colorMode()`

`brightness()`
Extracts the brightness value from a color

`color()`
Creates colors for storing in variables of the `color` datatype

`green()`
Extracts the green value from a color, scaled to match current `colorMode()`

`hue()`
Extracts the hue value from a color

`lerpColor()`
Calculates a `color` or `colors` between two `colors` at a specific increment

`red()`
Extracts the red value from a color, scaled to match current `colorMode()`

`saturation()`
Extracts the saturation value from a color

Setting

`background()`
Sets the color used for the background of the Processing window

`clear()`
Clears the pixels within a buffer

`colorMode()`
Changes the way Processing interprets color data

`fill()`
Sets the color used to fill shapes

Donate

`noStroke()`
Disables drawing the stroke (outline)

`stroke()`
Sets the color used to draw lines and borders around shapes

Lights Camera

Lights

`ambientLight()`
Adds an ambient light

`directionalLight()`
Adds a directional light

`lightFalloff()`
Sets the falloff rates for point lights, spot lights, and ambient lights

`lightSpecular()`
Sets the specular color for lights

`lights()`
Sets the default ambient light, directional light, falloff, and specular values

`noLights()`
Disable all lighting

`normal()`
Sets the current normal vector

`pointLight()`
Adds a point light

`spotLight()`
Adds a spot light

Donate

Material Properties

`ambient()`
Sets the ambient reflectance for shapes drawn to the screen

Sets the emissive color of the material used for drawing shapes drawn to the screen

`shininess()`
Sets the amount of gloss in the surface of shapes

`specular()`
Sets the specular color of the materials used for shapes drawn to the screen, which sets the color of highlights

Camera

`beginCamera()`
The `beginCamera()` and `endCamera()` functions enable advanced customization of the camera space

`camera()`
Sets the position of the camera

`endCamera()`
The `beginCamera()` and `endCamera()` functions enable advanced customization of the camera space

`ortho()`
Sets an orthographic projection and defines a parallel clipping volume

`perspective()`
Sets a perspective projection applying foreshortening, making distant objects appear smaller than closer ones

`frustum()`
Sets a perspective matrix defined through the parameters

`printCamera()`
Prints the current camera matrix to the Console (the text window at the bottom of Processing)

`printProjection()`
Prints the current projection matrix to the Console

Donate

Coordinates

`modelX()`
Returns the three-dimensional X, Y, Z position in model space

`modelY()`
Returns the three-dimensional X, Y, Z position in model space

`screenX()`

Takes a three-dimensional X, Y, Z position and returns the X value for where it will appear on a (two-dimensional) screen

`screenY()`

Takes a three-dimensional X, Y, Z position and returns the Y value for where it will appear on a (two-dimensional) screen

`screenZ()`

Takes a three-dimensional X, Y, Z position and returns the Z value for where it will appear on a (two-dimensional) screen

Transform

`applyMatrix()`

Multiplies the current matrix by the one specified through the parameters

`popMatrix()`

Pops the current transformation matrix off the matrix stack

`printMatrix()`

Prints the current matrix to the Console (the text window at the bottom of Processing)

`pushMatrix()`

Pushes the current transformation matrix onto the matrix stack

`resetMatrix()`

Replaces the current matrix with the identity matrix

`rotateX()`

Rotates a shape around the x-axis the amount specified by the `angle` parameter

`rotateY()`

Rotates a shape around the y-axis the amount specified by the `angle` parameter

`rotateZ()`

Rotates a shape around the z-axis the amount specified by the `angle` parameter

`rotate()`

Rotates a shape the amount specified by the `angle` parameter

Donate

`scale()`

shearX()
Shears a shape around the x-axis the amount specified by the `angle` parameter

shearY()
Shears a shape around the y-axis the amount specified by the `angle` parameter

translate()
Specifies an amount to displace objects within the display window

Structure

[] (array access)
The array access operator is used to specify a location within an array

= (assign)
Assigns a value to a variable

catch
The `catch` keyword is used with `try` to handle exceptions

class
Keyword used to indicate the declaration of a class

, (comma)
Separates parameters in function calls and elements during assignment

// (comment)
Explanatory notes embedded within the code

{} (curly braces)
Define the beginning and end of functions blocks and statement blocks such as the `for` and `if` structures

/** */ (doc comment)
Explanatory notes embedded within the code

. (dot)
Provides access to an object's methods and data

draw()
Called directly after `setup()` and continuously executes the lines of code contained ir `Donate` lock until the program is stopped or `noLoop()` is called

Search

`extends`

Allows a new class to *inherit* the methods and data fields (variables and constants) from an existing class

`false`

Reserved word representing the logical value "false"

`final`

Keyword used to state that a value, class, or method can't be changed

`implements`

Implements an *interface* or group of *interfaces*

`import`

The keyword `import` is used to load a library into a Processing sketch

`loop()`

Causes Processing to continuously execute the code within `draw()`

`/* */ (multiline comment)`

Explanatory notes embedded within the code

`new`

Creates a "new" object

`noLoop()`

Stops Processing from continuously executing the code within `draw()`

`null`

Special value used to signify the target is not a valid data element

`() (parentheses)`

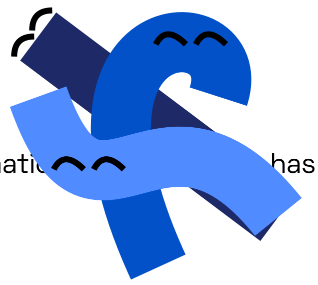
Grouping and containing expressions and parameters

`popStyle()`

Saves the current style settings and `popStyle()` restores the prior settings

`pop()`

The `pop()` function restores the previous drawing style settings and transformati has changed them



`private`

This keyword is used to disallow other classes access to the fields and methods withi

Donate

`public`

`pushStyle()`

Saves the current style settings and `popStyle()` restores the prior settings

`push()`

The `push()` function saves the current drawing style settings and transformations, while `pop()` restores these settings

`redraw()`

Executes the code within `draw()` one time

`return`

Keyword used to indicate the value to return from a function

`; (semicolon)`

A statement terminator which separates elements of the program

`setLocation()`

The `setLocation()` function defines the position of the Processing sketch in relation to the upper-left corner of the computer screen

`setResizable()`

By default, Processing sketches can't be resized

`setTitle()`

The `setTitle()` function defines the title to appear at the top of the sketch window

`setup()`

The `setup()` function is called once when the program starts

`static`

Keyword used to define a variable as a "class variable" and a method as a "class method

`super`

Keyword used to reference the superclass of a subclass

`this`

Refers to the current object (i

`thread()`

Launch a new thread and call the specified function from that new thread

`true`

Reserved word representing the logical value "true"

`try`

The `try` keyword is used with `catch` to handle exceptions

Donate

Keyword used to indicate that a function returns no value

Control

Conditionals

`break`

Ends the execution of a structure such as `switch`, `for`, or `while` and jumps to the next statement after

`case`

Denotes the different names to be evaluated with the parameter in the `switch` structure

`?: (conditional)`

A shortcut for writing an `if` and `else` structure

`continue`

When run inside of a `for` or `while`, it skips the remainder of the block and starts the next iteration

`default`

Keyword for defining the default condition of a `switch`

`else`

Extends the `if` structure allowing the program to choose between two or more blocks of code

`if`

Allows the program to make a decision about which code to execute

`switch`

Works like an `if else` structure, but `switch` is more convenient when you need to select between three or more alternatives

Relational Operators

`== (equality)`

Determines if two values are equivalent

`> (greater than)`

Tests if the value on the left is larger than the value on the right

`>= (greater than or equal to)`

Tests if the value on the left is larger than the value on the right or if the values are equivalent

Determines if one expression is not equivalent to another

`< (less than)`
Tests if the value on the left is smaller than the value on the right

`<= (less than or equal to)`
Tests if the value on the left is less than the value on the right or if the values are equivalent

Iteration

`for`
Controls a sequence of repetitions

`while`
Controls a sequence of repetitions

Logical Operators

`&& (logical AND)`
Compares two expressions and returns `true` only if both evaluate to `true`

`! (logical NOT)`
Inverts the Boolean value of an expression

`|| (logical OR)`
Compares two expressions and returns `true` if one or both evaluate to `true`

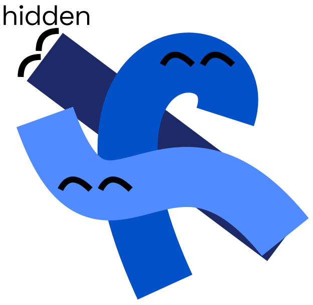
Environment

`cursor()`
Sets the cursor to a predefined symbol, an image, or makes it visible if already hidden

`delay()`
The `delay()` function causes the program to halt for a specified time

`displayDensity()`
Returns "2" if the screen is high-density and "1" if not

`displayHeight`
Variable that stores the height of the computer screen



Donate

variable that stores the width of the computer screen

`focused`

Confirms if a Processing program is ”focused”

`frameCount`

The system variable that contains the number of frames displayed since the program started

`frameRate`

The system variable that contains the approximate frame rate of the software as it executes

`height`

System variable which stores the height of the display window

`noCursor()`

Hides the cursor from view

`noSmooth()`

Draws all geometry and fonts with jagged (aliased) edges and images with hard edges between the pixels when enlarged rather than interpolating pixels

`pixelDensity()`

It makes it possible for Processing to render using all of the pixels on high resolutions screens

`pixelHeight`

The actual pixel height when using high resolution display

`pixelWidth`

The actual pixel width when using high resolution display

`fullScreen()`

Opens a sketch using the full size of the computer’s display

`frameRate()`

Specifies the number of frames to be displayed every second

`settings()`

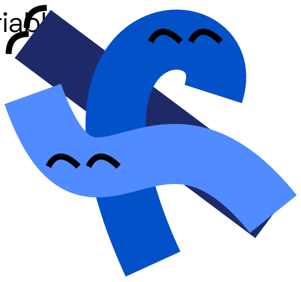
Used when absolutely necessary to define the parameters to `size()` with a variab

`size()`

Defines the dimension of the display window in units of pixels

`smooth()`

Draws all geometry with smooth (anti-aliased) edges



Donate

`width`

System variable which stores the width of the display window

Search

Contact Us

Feel free to write us!
foundation@processing.org

Twitter Medium Instagram GitHub

Processing is an open project initiated by Ben Fry and Casey Reas. It is developed by a team of volunteers around the world.

Donate