

Struktur

Programmablauf

setup()

Methode, wird einmalig zu Beginn des Programms ausgeführt. Dient zum Setzen von Anfangswerten, bevor der erste Aufruf von *draw()* erfolgt.

draw()

Methode, wird nach *setup()* immer wieder aufgerufen und ist für alle Bildschirmausgaben und für die Beantwortung von Ereignissen verantwortlich.

redraw()

führt den Code von *draw()* einmal aus. Damit ist es möglich, Bildschirmupdates nur dann durchzuführen, wenn sie tatsächlich nötig sind, etwa nach einem Tastatur- oder Mausereignis.

noLoop()

unterbindet den automatischen wiederholten Aufruf von *draw()*. Bei Verwendung sollte diese Anweisung in der letzten Zeile von *setup()* stehen. Beim Anstehen von Zeichenoperationen muss *redraw()* explizit aufgerufen werden, oder *draw()* mittels *loop()* wieder aktiviert werden.

loop()

aktiviert den automatischen Aufruf von *draw()*.

exit()

Methode, veranlasst das Ende des laufenden Programms, bei Ausführung einer *draw()*-Methode nach dem Beenden des aktuellen *draw()*. Ruft *stop()* auf.

stop()

Methode, wird bei Beendigung des Programmablaufs ausgeführt (falls sie vorhanden ist). Hier können z.B. erzeugte Audioverbindungen gelöst oder temporäre Dateien entfernt werden.

delay(milliseconds)

pausiert die Programmabarbeitung für *milliseconds* Millisekunden. Bei Angabe innerhalb von *draw()* wird erst nach seiner Beendigung pausiert.

import

Bindet Bibliotheken (Sammlungen von hilfreichen Klassen) in den Sketch ein. Mit einem * werden alle enthaltenen Klassen gewählt. Alternativ kann man in der IDE 'Import Library' im Menü verwenden.

Objektorientierung

class

Schlüsselwort zur Deklaration einer Klasse

```
public class Auto {  
  
    // Attribute  
    int baujahr;  
    string bezeichnung;  
  
    // ein Konstruktor mit 2 Parametern  
    public Auto(String name, int baujahr) {  
        bezeichnung = name;  
        this.baujahr = baujahr;  
    }  
  
    // Methoden  
    public void hupe() { ... }  
    void drehzahlanzeigen() { ... }  
  
}
```

public

gewährt anderen Klassen den Zugriff auf Attribute und Methoden. Default in Processing.

private

verbietet anderen Klassen den direkten Zugriff auf Attribute und Methoden.

static

macht eine Variable zum Klassenattribut, das allen abgeleiteten Objekten gemeinsam ist. Macht Methoden zu Klassenmethoden, deren Aufruf auch ohne Erzeugung eines Objekts möglich ist.

new

erzeugt ein neues Objekt aus einer Klasse

. (Punkt)

dient zur Referenzierung der Methoden und Attribute eines Objekts

```
Pattern p1 = new Pattern();  
p1.size = 100;  
p1.show("hello");
```

super

erlaubt den Zugriff auf die Superklasse einer Klasse

this

referenziert die aktuelle Klasse

extends

erlaubt die Vererbung von Methoden und Attributen aus einer bereits existierenden Klasse.

```
class BunterPunkt extends Punkt { ..... }
```

implements

dient als Ersatz der Mehrfachvererbung; die in Java nicht direkt möglich ist. Besonders für Threads nötig, die damit über das *Runnable*-Interface eine *run()*-Methode erhalten.

```
class Wackelpunkt extends Punkt implements Runnable { ..... }
```

Ausnahmen (exceptions)

try catch finally

erlaubt die Reaktion auf mögliche 'Fehler'-Fälle. Die Anweisungen im optionalen finally-Zweig werden in jedem Fall ausgeführt!

Speziell bei Dateizugriffen oder Netzwerkkommunikation kann man sauber programmieren und vermeidet viele verschachtelte if – else Zweige.

```
try {
    oeffneDatei();           // Fehler bei nicht existierender Datei
    schreibeDatei();        // Fehler z.B. bei fehlenden Schreibrechten
    anzahlBytes = ... ;
}
catch (IOException error) {
    anzahlBytes = -1;       // Versagen kennzeichnen
}
finally {
    schliesseDatei();      // Datei schließen und Lock freigeben
}
return anzahlBytes;       // Ergebnis zurückmelden
```

throw

erlaubt das gezielte Auslösen einer Exception.

Darstellung

size(width, height)

setzt Breite und Höhe des Grafikfensters in Pixel. Dies **muss** die erste Zeile in *setup()* sein. width und height sollten als Zahlen und nicht als Variable eingetragen werden, da sie für den Applet-Export fixiert sein müssen.

size(width, height, mode)

setzt Breite und Höhe des Grafikfensters in Pixel und bestimmt den Darstellungsmodus. Mögliche Werte für mode sind **P2D** (schnell, nicht so genau, für Pixeldaten geeignet.), **P3D** (schnelles 3D Rendering bei verringerter Qualität), **JAVA2D** (Standardeinstellung, hohe Qualität), **OPENGL** (Unterstützung der Hardwarebeschleunigung einer OpenGL-Grafikkarte), **PDF** (schreibt Grafik als Vektordaten in eine Datei, entsprechende Bibliothek importieren). Grafiken, die größer sind als der Bildschirm, können mithilfe von `createGraphics()` erzeugt werden.

Syntax

{ } (geschwungene Klammern, curly braces)

begrenzen einen Anweisungsblock, sowie Anfangswerte in der Definition von Arrays.

=

Zuweisung von Werten

; (Semikolon)

beendet eine Anweisung und trennt die Parameter einer *for*-Anweisung.

() (runde Klammern, parentheses)

beinhalten Parameter von Funktionen und Anweisungen, steuern die Berechnungsreihenfolge in mathematischen Ausdrücken.

[] (eckige Klammern, square brackets)

erlauben den Zugriff auf die Elemente eines Arrays.

```
v[7] = v[8]-3;
```

null

spezieller Datentyp, der ungültige (undefinierte) Werte kennzeichnet.

void

die Funktion besitzt keinen Rückgabewert

```
/* */
```

mehrzeiliger Kommentar

//

einzeiliger Kommentar

/ */ (doc comment)**

Ein JavaDoc-Kommentar. Der Textinhalt wird beim Export des Programms in die erzeugte *index.html* aufgenommen.

```
/**  
    Zeichnet ein Rechteck.  
    Gibt seinen Flächeninhalt als float zurück.  
*/
```

false

true

logische Wahrheitswerte

, (Komma)

trennt Parameter in Funktionsaufrufen und Listen

```
int a=2, b=4;  
int[] c = {10,20,30};  
line(x1,y1,x2,y2);
```

return

kennzeichnet den Rückgabewert einer Funktion und veranlasst die sofortige Rückkehr des Programmablaufs zum Aufrufer der Funktion.

final

macht eine Variable zur unveränderbaren Konstante. Von einer final-Klasse können keine Unterklassen erzeugt werden. Eine final-Methode kann nicht überschrieben werden.

Umgebung

screen

Objekt mit den Attributen **screen.width** und **screen.height**, sie geben die Bildschirmgröße (Desktopausmaße) in Pixel an.

Für bildschirmfüllende Präsentationen sollte man das Programm mittels Present starten, da bei Run ein Fensterrahmen sichtbar bleibt.

width

die aktuelle Fensterbreite

height

die aktuelle Fensterhöhe

frameRate(fps)

legt die maximale zu erreichende Anzahl von *draw()*-Aufrufen pro Sekunde fest. Der Standardwert beträgt 60. Sollte in *setup()* stehen.

frameRate

gibt die mittlere erreichte fps-Anzahl der *draw()*-Aufrufe an. Erst nach einigen *draw*-Zyklen sinnvoll abzufragen.

frameCount

gibt an, wie viele *draw()*-Aufrufe bisher stattgefunden haben. Innerhalb *setup()* ist der Wert 0.

focused

logischer Wert, gibt an, ob das Programm aktiv (fokussiert) ist und Maus- und Tastaturereignisse entgegennimmt.

online

logischer Wert, gibt an, ob das Programm innerhalb eines Browsers läuft.

noCursor()

versteckt den Mauscursor

cursor()

aktiviert den Mauscursor, falls er versteckt war.

cursor(mode)

bestimmt die Erscheinungsform des Mausursors. Mögliche Werte für *mode* sind **ARROW**, **CROSS**, **HAND**, **MOVE**, **TEXT** und **WAIT**.

cursor(img, x, y)

Der Mauscursor wird durch das *PImage img* ersetzt, dessen Hotspot bei (x/y) liegt.

Daten

Einfache

color

Datentyp zur Speicherung von Farbwerten.

| | |
|----------------|---|
| char | Unicode 16bit |
| boolean | false, true |
| byte | 8bit signed: -128...127 |
| short | 16bit signed: -32768..32767 |
| int | 32bit signed: +- 2 Milliarden |
| long | 64 bit signed: +- 19stellig |
| float | 32bit, -3.40282347E+38 ... 3.40282347E+38 |
| double | 64bit float mit mehr Nachkommastellen |

Um bei Zifferangaben long zu erzwingen wird der Buchstabe l an die Ziffernfolge gehängt. Analog wird f für float angehängt.

```
long z = 33l;      33 und Buchstabe l
float x = 3.2f;
double y = 234.01 + 5.231e-6;
```

Processing-Funktionen verwenden die Datentypen long und double nicht. Parameter sind demnach mit (int) oder (float) zu casten.

Zusammengesetzte

String

eine Zeichenfolge. Ihre direkte Angabe geschieht immer mittels doppelter Anführungszeichen (*chars* mit einfachen). Die Verwendung von Escape-Charactern ist möglich.

```
String wort1 = "Text";
char[] daten = {'T','e','x','t'}
String wort2 = new String(daten);
```

Array

eine Liste von Werten. Der erste Wert trägt den Index 0. Arrays werden als Objekte behandelt und müssen deshalb mittels *new* erzeugt werden. Jedes Array besitzt das Attribut **length**.

```
int[] feld = new int[2000];
```

Object

ein Auftreten (eine Instanz) einer Klasse. Während die Klasse das 'Modell' darstellt, ist das Objekt ihre tatsächliche Verkörperung.

```
class Hund { ..... } // Beschreibung der Eigenschaften und Fähigkeiten
Hund waldi = new Hund(); // ein konkreter Hund wird erzeugt
waldi.bell(); // Verhalten: Methode der Klasse
waldi.beine = 4; // Eigenschaft: ein Attribut wird gesetzt
```

Umwandlung

char(d)
int(d)

byte(d)
float(d)
boolean(d)

verwandelt einen einfachen Datentyp in das gewünschte Format. *boolean(0)* ergibt *false*, alle anderen Werte von *d* ergeben *true*.

str(d)

ergibt die Stringdarstellung eines einfachen Datentyps oder eines Arrays.

binary(d)

verwandelt *byte*, *char*, *int* oder *color* in einen String aus Binärwerten

unbinary(s)

wandelt die Stringdarstellung einer Binärzahl in einen *int*

hex(d)

verwandelt *byte*, *char*, *int* oder *color* in einen String aus Hex-Werten

unhex(s)

wandelt die Stringdarstellung einer Hexagesimalzahl in einen *int*

| |
|-------------------------|
| Stringfunktionen |
|-------------------------|

match(str, regExpr)

wendet eine RegularExpression auf den String an. Ergibt ein *StringArray*. Bei keinem Treffer wird *null* zurückgegeben.

split(str, delimiter)

trennt einen String an jedem Auftreten des *delimiter-characters* und ergibt ein Array von Strings

splitTokens(str)

splitTokens(str, tokens)

erlaubt das Auftrennen von Strings an mehreren Zeichen. Ohne Angabe von *tokens* wird jeder Whitespace als Trenner betrachtet.

```
String s = "a,b , , c";  
String[] p = splitTokens(s, ", "); // ergibt ["a","b","c"]
```

join(strArray, separator)

ergibt einen zusammengesetzten String.

trim(str)

trim(strArray)

entfernt Whitespace am Beginn und Ende von Strings

nf(intValue, digits) **nf(floatValue, left, right)**

number formatting: erzeugt formatierte Strings zur Darstellung der Zahlenwerte. Die Eingabe darf auch ein Array sein.

```
float a = 7.21;  
String aa = nf(a,3,5);  
print(aa);           // zeigt "007.21000"
```

nfc(...)

number formatting with commas: wie oben, es werden jedoch zwischen Tausendergruppen Kommata eingefügt.

nfs(...)

number formatting into strings: wie nf, doch mit einem führenden Leerzeichen bei positiven Werten, um eine gemeinsame spaltenweise Ausgabe mit negativen Zahlen zu erleichtern.

nfp(...)

number formatting with plus sign: wie nf, erzeugt jedoch immer ein führendes Vorzeichen + oder -.

Arrayfunktionen

shorten(array)

ergibt das array ohne sein letztes Element

concat(array1, array2)

ergibt die Aneinanderfügung der Arrays gleichen Typs

append(array, element)

ergibt das am Ende um *element* ergänzte Array

subset(array, index) **subset(array, index, length)**

ergibt ein Teil-Array ab *index*. Ohne Angabe von *length* bis zum Ende.

expand(array) **expand(array, length)**

verlängert ein Array auf *length* Elemente. Ohne Angabe von *length* wird die Länge verdoppelt.

sort(array) **sort(array, count)**

ergibt ein sortiertes Array. *count* erlaubt die Sortierung der ersten *count* Elemente alleine. Das ursprüngliche Array wird nicht verändert.

reverse(array)

ergibt das Array mit umgekehrter Reihenfolge der Elemente

arraycopy(src, dest)

arraycopy(src, dest, length)

arraycopy(src, srcindex, dest, destindex, length)

kopiert ein Array oder Teile in ein anderes Array und gibt das neue Array zurück. Ist effizienter als das Kopieren von Elementen mit Schleifen.

splice(array, value, index)

splice(array, array2, index)

fügt einen Wert oder ein zweites array ab einer Stelle in ein Array ein und gibt das neue Array zurück

Kontrolle

Vergleichsoperatoren

== != < > <= >=

gleich, ungleich, kleiner, größer, kleingleich, größergleich

logische Operatoren

&& || !

und, oder, not

Iterationen

for(init; test; update) {...}

for-Schleife

while (expression) {...}

while-Schleife

do {...} while (expression)

do-Schleife

Bedingungen

? :

```
result = condition ? exprTrue : exprFalse
a = (b<100) 10 : 20;           // Kurzform
if (b<100) { a=10; }         // äquivalente Langform
else { a=20; }
```

break

stoppt die Abarbeitung eines *switch*, *for* oder *while* -Blocks und springt zur ersten nachfolgenden Anweisung.

continue

stoppt die weitere Abarbeitung einer Schleife und beginnt die nächste Iteration.

if (condition) {...}
if (condition) {...} else {...}

if-Verzweigung

switch case default

Mehrfachabfrage. Der Typ der switch-Variable muss nach int konvertierbar sein (boolean, char, byte, short, int). Nach case müssen Konstante stehen, die schon zur Compilierzeit bekannt sind.

```
int num = 2;
switch (num) {
  case 0 : { ...; break; }
  case 2 : { ...; }
  default : { ...; break; }
}
```

Shapes

PShape

Datentyp für die Speicherung von Shapes. Momentan sind dies SVG-Shapes (Scalable Vector Graphics) wie sie von Inkscape oder Illustrator erzeugt werden. Noch wird nicht die vollständige Implementation unterstützt.

Attribute: `.width` , `.height`

Methoden:

- `.setVisible(boolean)` , `boolean isVisible()` - Sichtbarkeit
- `.disableStyle()` , `.enableStyle()` - Processing-Style oder Original-Style
- `.getChild("ID")` - ergibt Kindelement eines Shapes
- `.translate(x,y)` , `.translate(x,y,z)`
- `.rotate(angle)` - dreht um die linke obere Ecke
- `.rotateX(angle)` , `.rotateY(angle)` , `.rotateZ(angle)` - für P3D und OpenGL
- `.scale(amount)` , `.scale(amtX,amtY)` , `scale(amtX,amtY,amtZ)`

loadShape(filename)

lädt eine SVG Datei aus dem Datenverzeichnis des Sketches. Der Name kann auch eine URL auf dem gleichen Server wie der Sketch sein. Bei Misserfolg wird *null* zurückgegeben.

shape(sh,x,y)

shape(sh,x,y,width,height)

zeigt das SVG-Shape sh an. Die Bedeutung der 4 Koordinaten kann durch *shapeMode()* geändert werden.

shapeMode(MODE)

Der Default-Modus *CORNER* versteht die vier Koordinaten als Position der linken oberen Ecke, sowie waagrechte und senkrechte Größe. *CORNERS* erwartet die Koordinaten einer Ecke und die der Ecke gegenüber. *CENTER* verwendet die Mittelpunktskoordinaten, sowie Breite und Höhe.

pushStyle()

speichert alle aktuellen Stileinstellungen (stroke, fill, text, shapes, material) auf einen Stack.

popStyle()

holt die oberste gespeicherte Stileinstellung zurück und setzt alle entsprechenden Parameter. Aktuelle Werte werden überschrieben.

Ebene Primitive

point(x, y)

point(x, y, z)

zeichnet einen Punkt von 1 Pixel Größe. Für 3 Koordinaten muss in *size()* der Modus *P3D* oder *OPENGL* gesetzt worden sein. Die Koordinaten können *integer* oder *float* sein.

line(x1, y1, x2, y2)

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

zeichnet eine Strecke. Ihre Farbe wird mit *stroke()* gewählt, ihre Dicke mit *strokeWeight()*, sie hat kein Inneres.

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

zeichnet ein Dreieck

rect(x, y, width, height)

zeichnet ein Rechteck. Sein Bezugspunkt kann mit *rectMode()* bestimmt werden.

quad(x1, y1, x2, y2, x3, y3, x4, y4)

zeichnet ein allgemeines Viereck

ellipse(x, y, width, height)

zeichnet eine Ellipse. Ihr Bezugspunkt kann mit `ellipseMode()` bestimmt werden.

arc(x, y, width, height, start, stop)

zeichnet einen Bogen als Rand einer entsprechenden Ellipse. Anfangs- und Endwinkel je nach gewähltem Winkelmodus.

Kurven

curve(a1x,a1y, p1x,p1y, p2x,p2y, a2x,a2y)

curve(a1x,a1y,a1z, p1x,p1y,p1z, p2x,p2y,p2z, a2x,a2y,a2z)

zeichnet eine Splinekurve aus 2 Punkten und 2 Ankern

curvePoint(start,control1,control2,end,t)

wertet eine Kurve beim Parameter $0 \leq t \leq 1$ aus. Wird erst für x, dann für y ausgeführt, um die beide Koordinaten zu erhalten.

curveDetail(detail)

Setzt die Auflösung der Kurvendetails (Default 20). Nur in P3D oder OPENGL sinnvoll.

curveTangent(start, control1, control2, end, t)

berechnet die Tangente an einen Kurvenpunkt.

curveTightness(squishy)

bestimmt, wie die Kurve zwischen den Vertexpunkten verläuft. `squishy=0.0` für Spline, `1.0` für gerade Teilstrecken. `-5.0` bis `5.0` verformt die Kurve stärker.

bezier(a1x,a1y, cp1x,cp1y, cp2x,cp2y, a2x,a2y)

bezier(a1x,a1y,a1z, cp1x,cp1y,cp1z, cp2x,cp2y,cp2z, a2x,a2y,a2z)

eine Bezierkurve aus den gegebenen Anker- und Kontrollpunkten.

bezierPoint(start, control1, control2, end, t)

wertet die Bezierkurve am Punkt t aus

bezierDetail(detail)

Setzt die Auflösung der Kurve (Default 20). Nur in P3D oder OPENGL sinnvoll.

bezierTangent(start, control1, control2, end, t)

berechnet die Tangente an einem Punkt der Bezierkurve

räumliche Primitive

box(size)

box(width, height, depth)

zeichnet einen Würfel oder einen Quader

sphere(radius)

zeichnet eine Kugel

sphereDetail(detail)

bestimmt die Auflösung der folgenden Spheres, Default ist 30, was $360/30 = 12^\circ$ entspricht. Bei vielen kleinen Kugeln pro Frame kann durch eine Verringerung die Anzeigegeschwindigkeit erhöht werden.

Attribute

smooth()

noSmooth()

schaltet Antialiasing ein oder aus

strokeWeight(width)

Setzt die Breite der Umrandung von Shapes in Pixeln. Nicht für P3D.

strokeCap(MODE)

bestimmt die Form von Linienenden. **SQUARE** rechteckig bis zu den Endpunkten, **PROJECT** rechteckig über die Eckpunkte erweitert, **ROUND** (default) mit angesetzten abgerundeten Enden.

strokeJoin(MODE)

bestimmt die Form der Liniensegmentenden. **MITER** (default) ergibt spitze Ecken, **BEVEL** abgeschrägte, **ROUND** gerundete. Nicht für P2D, P3D, OPENGL.

rectMode(MODE)

bestimmt die Position des Bezugspunktes der Rechtecksdefinition $rect(a,b,c,d)$. **CORNER** (default) versteht a,b als linke obere Ecke und c,d als Breite und Höhe, **CORNERS** versteht a,b und c,d als gegenüberliegende Eckpunkte. **CENTER** bezieht sich auf den Mittelpunkt a,b und Größe c,d, **RADIUS** auf den Mittelpunkt a,b und Halbmesser c und d.

ellipseMode(MODE)

wie *rectMode*. Default ist **CENTER**.

Vertex

vertex(x, y)

vertex(x, y, z)

vertex(x, y, u, v)

vertex(x, y, z, u, v)

bestimmen einen Vertex-Punkt in einem Shape. Mit u,v kann das Texture-Mapping gesteuert werden.

curveVertex(x, y)

curveVertex(x, y, z)

erklärt einen Punkt eines Spline-Shapes. Mindestens 4 müssen im Shape definiert werden, um eine Kurve zwischen dem zweiten und dritten zeichnen zu können.

bezierVertex(cp1x,cp1y, cp2x,cp2y, ax,ay)

bezierVertex(cp1x,cp1y,cp1z, p2x,cp2y,cp2z, ax,ay,az)

bestimmt die Koordinaten von zwei Kontrollpunkten und einem Ankerpunkt

beginShape(MODE)

beginnt die Definition eines Shapes. Vertexfestlegungen müssen folgen.

endShape()

endShape(CLOSE)

beendet die Vertexliste als offenes oder automatisch geschlossenes Shape.

texture(image)

legt innerhalb einer Shapedefinition die Textur in Form eines *PImage* fest.

textureMode(MODE)

erklärt den Koordinatenraum für das Texture-Mapping. **IMAGE** (default) wählt die Pixelgröße des Bildes, **NORMALIZED** Werte von 0 bis 1.

Input

Maus

mouseX

mouseY

die aktuellen Mauskoordinaten

pmouseX **pmouseY**

die Mauskoordinaten aus dem vorangegangenen Frame

mouseButton

mögliche Ergebnisse sind die Systemkonstanten **LEFT**, **RIGHT**, **CENTER**.

mousePressed

logischer Wert, ob ein Mausbutton gedrückt ist. *mouseButton* gibt dann an, welcher es war.

mousePressed()

Funktion, die bei jeder Mausbuttonbetätigung aufgerufen wird.

mouseReleased()

Funktion, die bei jedem Loslassen eines Buttons aufgerufen wird.

mouseClicked()

Funktion, die nach jedem Click (nach Release) aufgerufen wird.

mouseMoved()

Funktion, die nach jeder Mausbewegung ohne Click aufgerufen wird.

mouseDragged()

Funktion, die nach jeder Mausbewegung mit Click aufgerufen wird.

Tastatur

key

der Wert der zuletzt gedrückten bzw. losgelassenen Taste. Für ASCII-Tasten. Kann den Wert **CODED** für Spezialtasten liefern.

keyCode

für **UP**, **DOWN**, **LEFT**, **RIGHT** Cursor, **ALT**, **CONTROL**, **SHIFT**. Zuerst muss überprüft werden, ob der Key **CODED** ist.

keyPressed

logischer Wert, ob eine Taste gedrückt ist.

keyPressed()

Funktion, die nach jedem Tastendruck aufgerufen wird.

```

void keyPressed() {
  if (key == CODED) {
    if (keyCode == UP) { dosomething(); }
    else if (keyCode == DOWN) { dosomethingelse(); }
  }
  else { dontdoanything(); }
}

```

keyReleased()

Funktion, die nach jedem Tastenloslassen aufgerufen wird.

keyTyped()

Funktion, die nach jedem Tastendruck aufgerufen wird. Keine Spezialtasten wie ctrl, shift, alt. Achtung: Autorepeat erzeugt mehrfache Aufrufe!

```

void draw() {} // damit das Programm läuft
void keyPressed() {
  println("pressed "+int(key)+" "+keyCode);
}
void keyReleased() {
  println("released "+int(key)+" "+keyCode);
}
void keyTyped() {
  println("typed "+int(key)+" "+keyCode);
}

```

Dateien

createInput(filename)

Ein Java Input-Stream wird im Datenordner erzeugt. Im Fehlerfall wird **null** zurückgegeben.

!!! Endet der Dateiname mit '.gz', wird automatisch entpackt. Wünscht man dies nicht, verwendet man stattdessen ***createInputRaw(filename)***.

selectInput(prompt)

öffnet ein Fenster zur Dateiauswahl. Ergibt den vollen Pfad zum gewählten Verzeichnis oder **null**, wenn nichts gewählt wurde. Das optionale prompt steht im Dateiwahlfenster.

selectFolder(prompt)

öffnet ein Fenster zur Dateiauswahl. Ergibt einen vollen Pfad der gewählten Datei als String oder **null**, wenn nichts gewählt wurde. Das optionale prompt steht im Dateiwahlfenster.

open(filename)

führt eine Datei aus

open(arguments)

übergibt einen String oder ein String-Array zur Ausführung an die Kommandozeile. Mit dem Array sind Parameter an ein Programm übergebbar.

```
String[] params = {"/usr/bin/vim","-help"};  
open(params);
```

[openStream(name)]

!!! wurde ersetzt durch createInput !!!

loadStrings(filename)

liest eine Datei aus dem 'data'-Verzeichnis als StringArray

loadBytes(filename)

liest eine Datei als ByteArray aus 'data'

BufferedReader

Objekt, um eine Datei zeilenweise zu lesen. UTF-8 Kodierung

createReader(filename)

erzeugt einen BufferedReader zum zeilenweisen Lesen von Strings. Diese Funktion ist das Gegenstück zu *createWriter()*.

Web

status(text)

zeigt einen Text in der Statuszeile des Browsers

link(url)

link(url ,target)

Verlinkt auf eine Seite im aktuellen oder einem neuen Fenster. Der volle Verweis "http://..." muss angegeben werden

param(name)

liest den Wert eines Parameters aus der laufenden Web-Browser-Seite als String ein.

Zeit und Datum

year()

month()

day()

hour()
minute()
second()

ergibt die aktuellen Werte der Systemuhr

millis()

ergibt die Zeit seit dem Programmstart in Millisekunden. Wichtig für das Timing von Animationssequenzen

Ausgabe

Text

print(...)
println(...)

schreibt Werte (auch Arrays) in die Processing-Konsole. Einzelne Werte sind mit + verkettbar.

Image

save(filename)

speichert ein Einzelbild in den Sketch-Folder. **tif, tga, jpg, png** sind möglich. Nicht im Browser.

saveFrame()
saveFrame("filename-###.ext")

speichert einen Einzelframe als fortlaufend nummeriertes Bild in den Sketch-Folder.

Dateien

createOutput(filename)

Ein Output-Stream wird im Datenordner erzeugt. Nötigenfalls werden Unterordner automatisch erzeugt. Im Fehlerfall wird **null** zurückgegeben. !!! Endet der Dateiname mit '.gz' wird automatisch eine GZIP-Datei erstellt.

selectOutput(prompt)

öffnet ein Fenster zur Dateiauswahl. Ergibt den vollen Pfad der gewählten Datei als String oder **null**, wenn nichts gewählt wurde. Existierende Dateien werden überschrieben. Das optionale prompt steht im Dateiwahlfenster.

saveStrings(filename, stringarray)

speichert zeilenweise, im Browser nicht möglich

saveBytes(filename, bytearray)

speichert Bytefolge

saveStream(destination, source)

speichert den Inhalt eines Streams. Destination kann ein file oder ein Filename sein. Ist im Wesentlichen eine effizientere Version der Anweisung **saveBytes(blahblah, loadBytes())**

createWriter(filename)

erzeugt eine neue Datei im Sketch-Ordner mit eigenem *PrintWriter* Objekt. UTF-8 encoding.

PrintWriter

Objekt für *createWriter*, Methoden **print()** , **println()** , **flush()** und **close()**.

beginRaw(renderer, filename)

öffnet eine Datei und lenkt alle folgenden Zeichenbefehle auf sie. Für **PDF** oder **DXF**-Renderer. Erzeugt 3D Vektordaten. Zur Qualitätsverbesserung kann **hint(ENABLE_DEPTH_SORT)** verwendet werden.

endRaw()

beendet die Ausgabe und schließt die Datei.

beginRecord(renderer, filename)

öffnet eine Datei und lenkt alle folgenden Zeichenbefehle auf sie UND ins Anzeigefenster des Programms. Für **PDF** oder **DXF**-Renderer

endRecord()

beendet die Ausgabe und schließt die Datei.

Transformationen

rotate(angle)

rotiert die folgenden Objekte um *angle* Radianen im Uhrzeigersinn um den Ursprung.

translate(x, y)

translate(x, y, z)

verschiebt die folgenden Objekte

shearX(angle) **shearY(angle)**

übt eine Scherung um den Ursprung in Uhrzeigerrichtung aus.
In manchen Dokumentationen durch *skewX()* und *skewY* ersetzt.

scale(size) **scale(sx, sy)** **scale(sx, sy, sz)**

skaliert die folgenden Objekte. Zu Beginn eines draw() wird automatisch auf 1 gesetzt.

rotateX(angle) **rotateY(angle)** **rotateZ(angle)**

für P3D und OPENGL, positive Winkel wirken gegen den Uhrzeiger.

resetMatrix()

ersetzt die aktuelle Transformationsmatrix durch die Identität

pushMatrix()

pusht die aktuelle Matrix auf den Matrixstack

popMatrix()

poppt eine Matrix vom Matrixstack

printMatrix()

schreibt die aktuelle Matrix ins die Textfenster

applyMatrix(n₀₀, n₀₁, ..., n₁₅)

multipliziert die aktuelle Transformationsmatrix mit der hier angegebenen.

Farbe

setzen

colorMode(mode) **colorMode(mode, range)** **colorMode(mode, range1, range2, range3)** **colorMode(mode, range1, range2, range3, range4)**

mode = **RGB** oder **HSB**, range ist der Wert für volle farbe, je nach Modell.
range für Graustufen, range1 bis 3 für rgb bzw. hsb, 4 für Alpha.

Default: 255

background(gray)
background(gray,alpha)
background(v1,v2,v3)
background(v1,v2,v3,alpha)
background(color)
background(color,alpha)
background(hex)
background(hex,alpha)

setzt die Hintergrundfarbe und ihren Alpha-Wert (diesen allerdings nicht im Hauptzeichenfenster). v1: red oder hue, v2: green oder saturation, v3: blue oder brightness. color: ein *Color-Datentyp*, auch hex: #FFAABB oder 0xFFCC0080 möglich

stroke(params)

Parameter wie *background*. Setzt die Farbe von Linien und Rändern einer Figur

fill(params)

Parameter wie *background*. Setzt die Farbe des Inneren von Figuren.

noStroke()

zeichnet keine Linie bzw. keinen Rand

NoFill()

das Innere der Figur wird nicht gefüllt und erscheint transparent.

Erzeugen und lesen

color(params)

erzeugt einen Farbwert. Parameter wie *background()*.

red(color) green(color) blue(color)
hue(color) brightness(color) saturation(color)
alpha(color)

extrahiert die gewünschten Farbanteile. Diese können zur Geschwindigkeitsoptimierung im colorMode mit 256 Werten auch über Bitshifts extrahiert werden.

```
float r1 = red(myColor);  
float g1 = green(myColor);  
float b1 = blue(myColor);
```

```
float r2 = myColor >> 16 & 0xFF;
float g2 = myColor >> 8 & 0xFF;
float b2 = myColor & 0xFF;
```

blendColor(color1, color2)

erzeugt je nach Farbmodell die Mischfarbe aus zwei gegebenen Farben.

lerpColor(color1, color2, amount)

ergibt die Zwischenfarbe zum Parameter $0 \leq amount \leq 1$

Image

PImage

Datentyp zur Speicherung von Bildern. GIF, JPG, TGA, PNG sind erlaubt.

Felder:

PImage.width Bildbreite

PImage.height Bildhöhe

PImage.pixels[] Pixelfarben als Array

Methoden

PImage.get() *.set()* *.copy()* *.mask()* *.blend()* *.filter()* *.save()*

PImage()

PImage(width, height)

PImage(java_awt_image)

erzeugt ein PImage-Objekt

createImage(width, height, format)

Erzeugt ein neues PImage Objekt. Größe in Pixel, format ist **RGB**, **ARGB** oder **ALPHA** (Graukanal Alpha)

Laden und anzeigen

loadImage(filename)

lädt ein Bild aus dem 'data'-Verzeichnis in ein PImage. Solche Ladeanweisungen sollten in *setup()* stehen, um *draw()* nicht zu verlangsamen. Der Name kann auch eine URL sein, bei Internetanwendungen muss das Bild vom gleichen Server stammen (außer das Applet wird signiert).

Bei Misserfolg des Ladens wird **null** zurückgegeben.

requestImage(filename)

lädt ein Bild aus dem 'data'-Verzeichnis in ein PImage. Dies geschieht in einem eigenen Thread, wodurch der Ablauf von *setup()* nicht durch das Abwarten des Ladevorgangs verzögert wird.

Solange das Bild nicht komplett eingelesen wurde, sind seine Attribute width

und height gleich 0. Konnte das Bild nicht geladen werden, sind die Attribute -1.

image(img, x, y)

image(img, x, y, width, height)

zeigt ein image an. Ohne Angabe der Breite und der Höhe wird die Originalgröße verwendet.

imageMode(MODE)

CORNER (default): die angegebenen Koordinaten x,y sind die linke obere Ecke, **CORNERS**: die Koordinaten x,y sind eine Ecke, $width,height$ werden als gegenüberliegende Ecke interpretiert.

tint(params)

Parameter wie bei *background*. Erlaubt die Angabe der Färbung und der Transparenz eines Bildes. *tint(255,128)* bedeutet halbdurchsichtig (außer *colorMode()* wurde verändert). Die Methode kann auch zur Färbung von 3D-Texturen verwendet werden.

noTint()

Das Bild behält seine ursprüngliche Färbung.

Pixels

get()

get(x, y)

get(x, y, width, height)

liest einen Pixelwert oder einen ganzen Block. Ohne Parameter wird der gesamte Bereich geladen. Langsamer als der direkte Zugriff über *pixels[]*. *get(x,y)* entspricht *pixels[y*width+x]*

set(x, y, color)

set(x, y, image)

setzt die Farbe oder ein Bild ein.

loadPixels()

erzeugt das Array *pixels[]*

pixels[]

eindimensionales Array der Pixelfarben vom *color*-Typ. Steht erst nach *loadPixels()* zur Verfügung.

updatePixels()

schreibt *pixels[]* ins Anzeigefenster zurück.

**copy(x, y, width, height, destx, desty,
destwidth, destheight)**
**copy(img, x, y, width, height, destx, desty,
destwidth, destheight)**

kopiert eine Pixelregion in eine andere oder einen Bildbereich in eine Pixelregion. Dabei sind Skalierungen möglich.

filter(MODE, level)

filtert das Anzeigefenster.

MODE:

THRESHOLD: Schwarzweiß nach Schwellwert **level** zwischen 0 (schwarz) und 1 (weiß), default ist 0.5.,

GRAY: in Graustufen,

INVERT: invertiert

POSTERIZE: verringert die Farbzahl auf level Farben

BLUR: Gauss-Weichzeichner mit level, Default Radius 1.

OPAQUE: setzt den Alphakanal auf deckend

ERODE: dunkelt helle Stellen nach level-Wert

DILATE: hellt dunkle Stellen nach level-Wert auf.

blend(params,MODE)

mischt eine Bildregion in eine zweite. *params* wie bei *copy()*, MODE kann **BLEND, ADD, SUBTRACT, LIGHTEST, DARKEST** sein.

Typographie

PFont

die Fontklasse. Um sie zu erzeugen, muss 'Create Font' im Tools Menü der Processing IDE ausgeführt werden. Dabei werden die Zeichen von der Vektordefinition in ein Bild gewandelt und im 'data'-Verzeichnis als .vlw-Datei abgelegt. Die Klassenmethode **.list()** liefert eine Liste der für Java sichtbaren installierten Schriftarten.

```
String[] fontList = PFont.list();  
println(fontList);
```

PFont()

PFont(inputStream)

Konstruktor.

Laden und anzeigen

createFont(name, size)
createFont(name, size, smooth)
createFont(name, size, smooth, charset)

Erzeugt einen PFont dynamisch aus einer Font-Datei (.ttf, .otf) im 'data'-Verzeichnis (oder installiert im Betriebssystem), falls der Font noch nicht konvertiert wurde. **smooth**=true für Antialiasing, **charset** ist ein char[] Array mit den zu erzeugenden Zeichen.
Funktioniert dynamisch so, wie die Wahl des 'Create Font' Tools der IDE. Der Default-Renderer JAVA2D greift wenn möglich auf die Originalschriftart zu und ignoriert die Bilder. P2D, P3D und OpenGL verwenden immer die Bilder (großer Tempogewinn).

loadFont(name)

lädt einen erzeugten Font in ein Pfont Objekt.

textFont(font)
textFont(font, size)

aktiviert den zu verwendenden Font

text(data, x, y)
text(data, x, y, z)
text(stringdata, x, y, width, height)
text(stringdata, x, y, width, height, z)

zeichnet Text auf den Schirm. *data* kann String, Char, Int oder Float sein. *width* und *height* sind die Größe der Textbox.

| |
|------------------|
| Attribute |
|------------------|

textMode(MODE)

MODEL (default) transformierbar, mit Texture-Outlines. **SHAPE** mit Glyph-Outlines, nur PDF und OPENGL. **SCREEN** zeichnet direkt in den Vordergrund (schnell für P2D und P3D).

textSize(size)

Fontgröße in Pixel

textAlign(align)
textAlign(align, yalign)

bestimmt die Ausrichtung. **align** kann **LEFT**, **CENTER**, **RIGHT** sein **yalign** kann **TOP**, **BOTTOM**, **CENTER**, **BASELINE** sein.

textLeading(distance)

Abstand zwischen einzelnen Texten in Pixels.

textWidth(data)

berechnet die Breite eines Char oder Strings

Metrik

textDescent()

liefert die Größe der Unterlängen (p tiefer als x)

textAscent()

liefert die Oberlänge (t höher als x)

Mathematik

+ - * / % ++ -- += -= *= /= %=

Rechenoperatoren. Beachte den Unterschied Prä- und Post-Inkrement und -Dekrement.

& | ! ^ << >>

bitweises AND, OR, NOT, XOR, Linksshift, Rechtsshift

PVector()

PVector(x,y)

PVector(x,y,z)

Klasse zum Umgang mit 2 oder 3-dimensionalen Vektoren
Beachte, dass Du etwa die Vektoraddition sowohl in der Form
`v3=v1.add(v2)` schreiben kannst, als auch `v3=PVector.add(v1,v2)`

Attribute: `.x` , `.y` , `.z`

Methoden:

| | |
|---|---|
| <code>.set(v)</code> , <code>.get()</code> | - Koordinaten setzen und lesen |
| <code>.mag()</code> | - Länge bestimmen |
| <code>.add(v)</code> , <code>.sub(v)</code> | - plus, minus |
| <code>.mult(c)</code> , <code>.mult(v)</code> | - Skalar- und Vektorprodukt |
| <code>.div(c)</code> , <code>.div(v)</code> | - Division durch Skalar oder Vektor |
| <code>.dist(v)</code> | - euklidischer Abstand |
| <code>.dot(v)</code> , <code>.cross(v)</code> | - Skalar- und Vektorprodukt |
| <code>.normalize()</code> | - ergibt den Einheitsvektor |
| <code>.limit(c)</code> | - skaliert auf Länge c |
| <code>.angleBetween(v)</code> | - Winkel |
| <code>.array()</code> | - Vektorobjekt in ein Float Array umwandeln |

Berechnungen

min(value1, value2)
min(value1, value2, value3)
min(array)

ergibt das Minimum von 2 oder 3 Zahlenwerte, bzw. das eines Zahlenarrays.

max(value1, value2)
max(value1, value2, value3)
max(array)

ergibt das Maximum von 2 oder 3 Zahlenwerte, bzw. das eines Zahlenarrays.

round(value)

rundet zur nächstgelegenen ganzen Zahl, ergibt einen Integer.

floor(value)

rundet zur nächstgelegenen ganzen Zahl ab, ergibt einen Integer.

ceil(value)

rundet zur nächstgelegenen ganzen Zahl auf, ergibt einen Integer.

abs(value)

ergibt den Absolutbetrag einer Zahl

mag(x, y)
mag(x, y, z)

ergibt die Länge des Vektors

dist(x1, y1, x2, y2)
dist(x1, y1, z1, x2, y2, z2)

ergibt die Entfernung der beiden Punkte

constrain(value, low, high)

beschränkt den Wert auf das gegebene Intervall. Ist er unter *low*, wird *low* zurückgegeben, ist er über *high*, wird *high* zurückgegeben, sonst bleibt er unverändert.

norm(value, low, high)

ergibt den Anteil des Wertes zwischen *low* und *high* als Zahl zwischen 0 und 1. Werte außerhalb des Intervalls werden nicht verändert.

```
norm(6,1,11); // ergibt 0.5, der Wert 6 liegt in der Mitte  
norm(2,1,5); // ergibt 0.25, der Wert 2 ist 25% von 1 Richtung 5 entfernt
```

map(value, low1, high1, low2, high2)

bestimmt den Anteil des Wertes im Intervall [*low1*,*high1*] und bildet ihn auf das Intervall [*low2*,*high2*] ab.

lerp(value1, value2, amount)

ergibt den linear interpolierten Wert von *value1* (amount=0) und *value2* (amount=1). $0.0 \leq \text{amount} \leq 1.0$

sq(value)

ergibt das Quadrat der Zahl

sqrt(value)

ergibt die Quadratwurzel der Zahl

pow(base, exponent)

ergibt die Potenzfunktion $\text{base}^{\text{exponent}}$.

exp(value)

ergibt die Exponentialfunktion der Zahl

log(value)

ergibt den natürlichen Logarithmus der Zahl

Trigonometrie

die trigonometrischen Funktionen werden in Radianten berechnet

degrees(rad_angle)

ergibt den Winkel in Grad

radians(deg_angle)

ergibt den Winkel in Radianten

sin(angle)

cos(angle)

tan(angle)

die üblichen Winkelfunktionen

asin(value)

acos(value)

inverse Sinus- und Cosinusfunktion. Die Rückgabewerte sind im Intervall von 0 bis π .

atan(value)

inverser Tangens, Der Rückgabewert ist im Intervall $-\pi/2$ bis $\pi/2$

atan2(y, x)

ergibt den Winkel der Richtung (x/y) mit der positiven x-Achse. Der

Rückgabewert ist im Intervall $-\pi$ bis π

Zufall

randomSeed(intvalue)

Startwert für den Zufallsgenerator

random(high)

random(low,high)

ergibt eine float-Zufallszahl von in $[0, high)$ bzw. $[low, high)$

noiseSeed(intvalue)

Startwert für den Rauschgenerator

noiseDetail(octaves, falloff)

Anzahl der Oktaven (Default 4) und Abfallfaktor (Default 0.5) für den Rauschgenerator

noise(x)

noise(x, y)

noise(x, y, z)

ergibt den Rauschwert eines ein-, zwei-, oder dreidimensionalen Perlin-Generators am angegebenen Ort.

Konstante

PI

QUARTER_PI

HALF_PI

TWO_PI

vordefinierte float-Zahlenwerte

Rendering

PGraphics

Grafikkontext für einen Offscreen Grafikpuffer. Wird mit *createGraphics()* erzeugt, mit *beginDraw()* initialisiert und mit *endDraw()* finalisiert.

Details unter <http://dev.processing.org/reference/core/>

createGraphics(width, height,renderer) **createGraphics(width, height,renderer,filename)**

erzeugt ein neues PGraphics Objekt P2D, P3D, JAVA2D. Wird als Buffer Offscreen-Grafik verwendet. Nicht für OPENGL (erlaubt keine Offscreen-Verwendung). DXF und PDF benötigen einen Dateinamen.

1bit-Transparenz für die Zeichenfläche ist vorhanden und bleibt bei *save()* in eine PNG oder TGA Datei erhalten.

!!! P2D ist noch nicht implementiert.

hint(HINTING)

spezielle technische Einstellungen für den aktuellen Renderer.

Licht und Kamera

Licht

-
-

Kamera

-
-

Koordinaten

-
-

Materialeigenschaften

-
-

Abweichungen von der Processing-Doc 1.2.1

- pushStyle() und popStyle() beschreibe ich unter 'Shapes', nicht unter 'Structure'