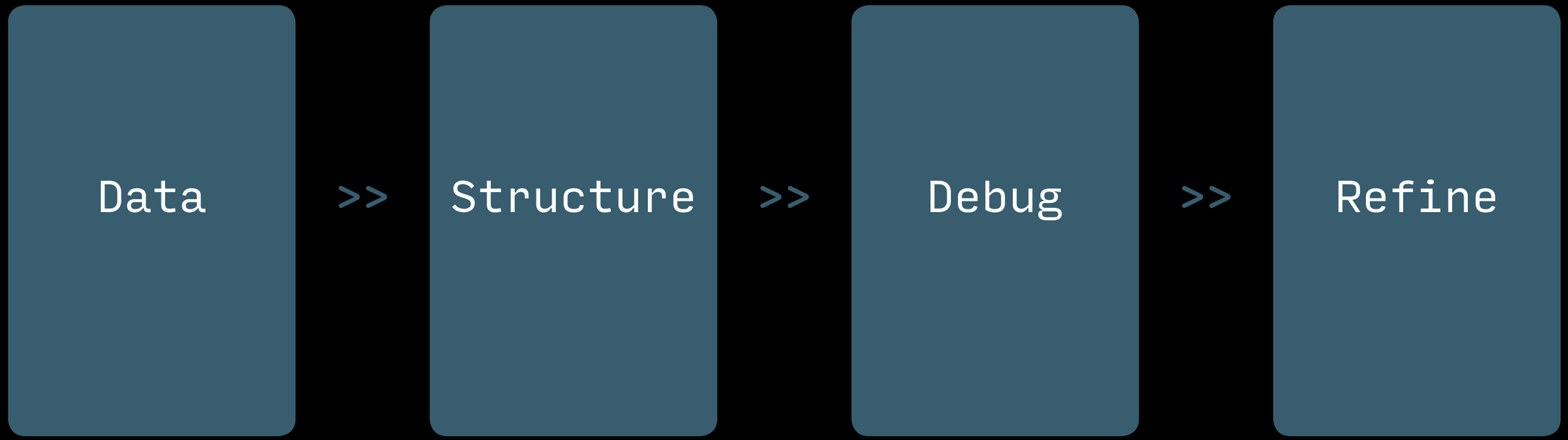
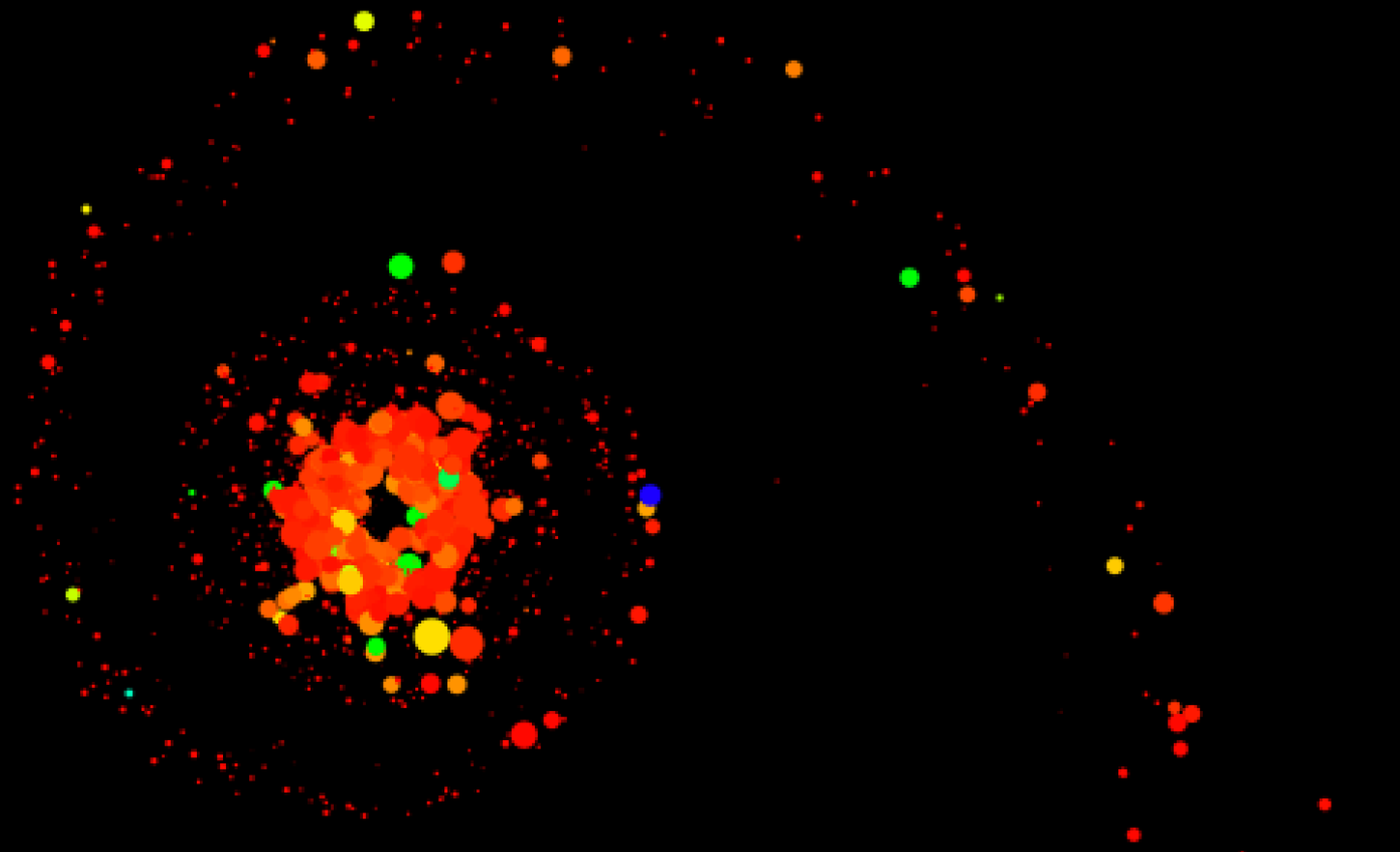
KREATIVES PROGRAMMIEREN

AN INFOVIS TOOLKIT FOR PROCESSING

Processing ToolKit



Processing ToolKit



Data

CSV

Structure

Radial

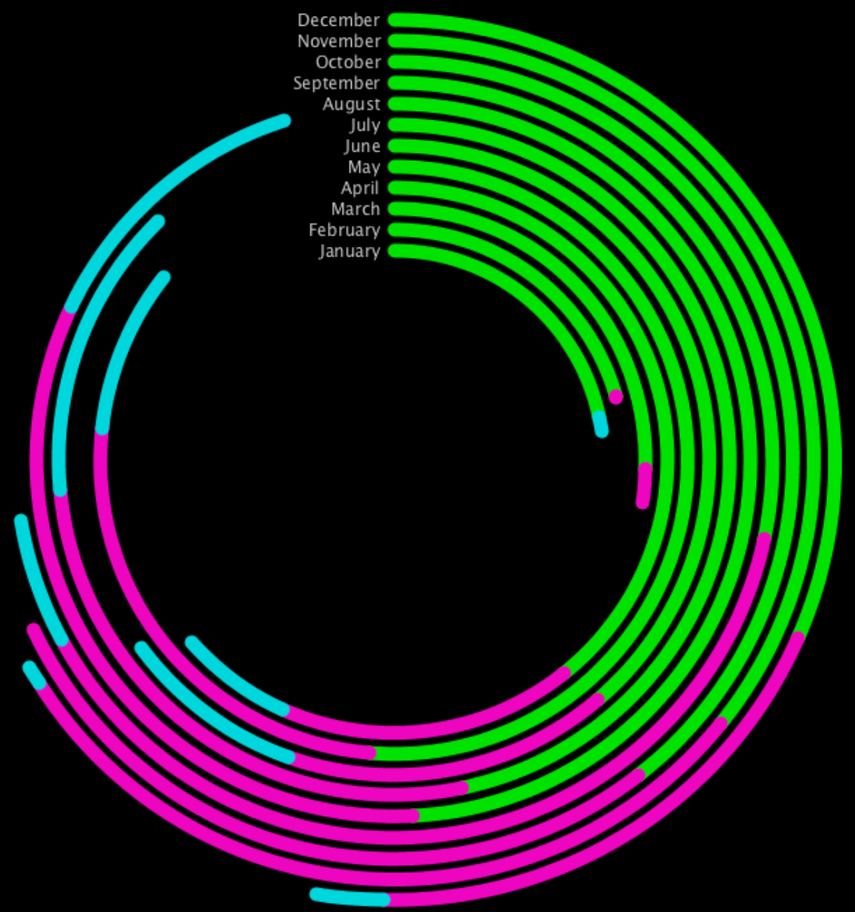
Debug

Framerate

Refine

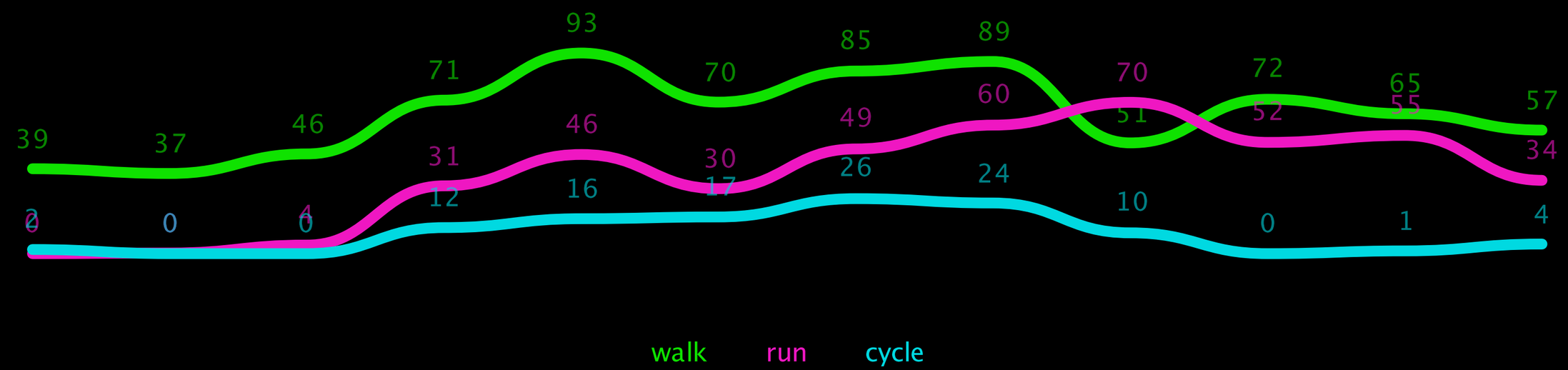
HSB

Processing ToolKit



Data	Structure	Debug	Refine
CSV	Concentric	mouseX	arc

Processing ToolKit



Data

CSV

Structure

Bezier

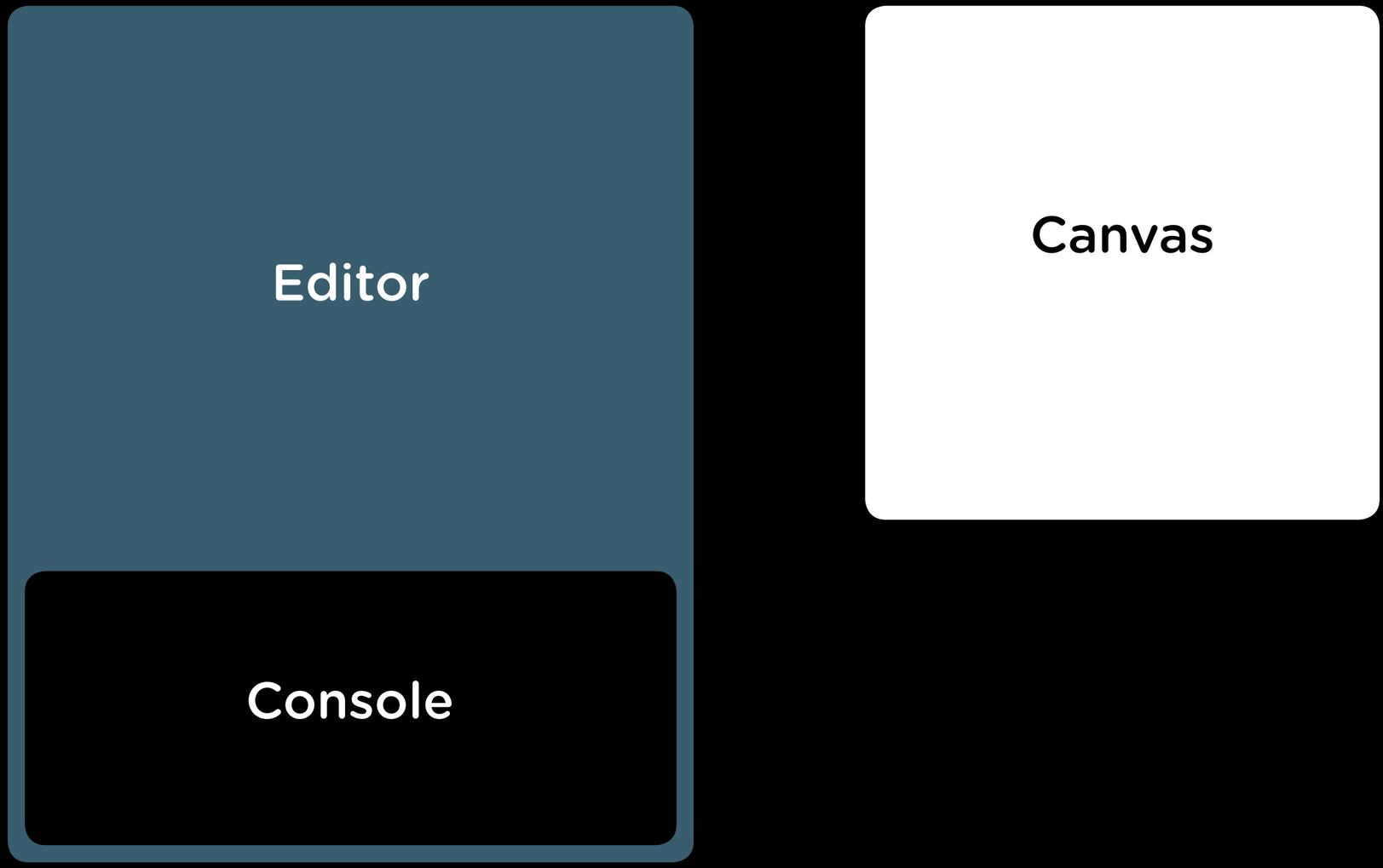
Debug

Annotate

Refine

Save PDF

Processing Elements



Code Structure

Global Context

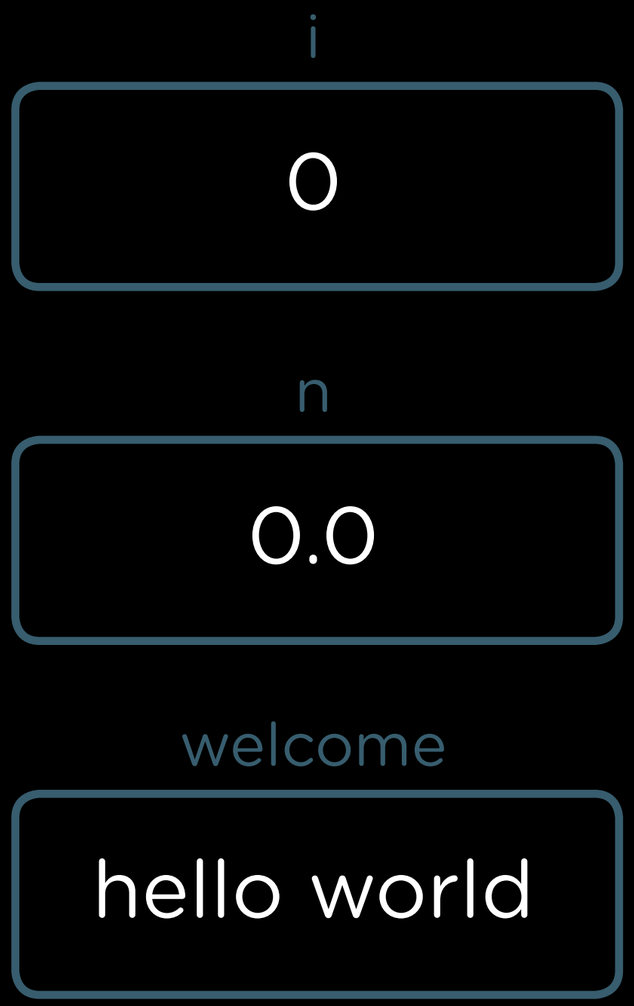
Setup

Draw loop

```
// variables, libraries

setup() {
  // code runs once
}

draw() {
  // code repeats
}
```

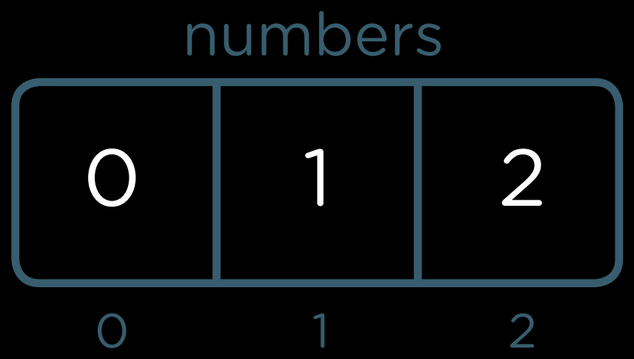
Editor

Basic Variables

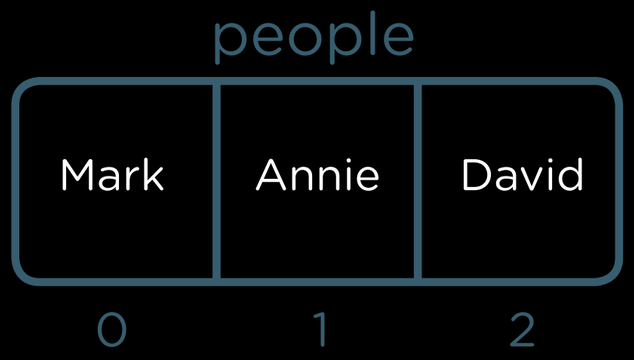



```
int i = 0;
// println(i);

float n = 0.0;
// println(n);

String welcome = "hello world"
// println(welcome);
```

Arrays

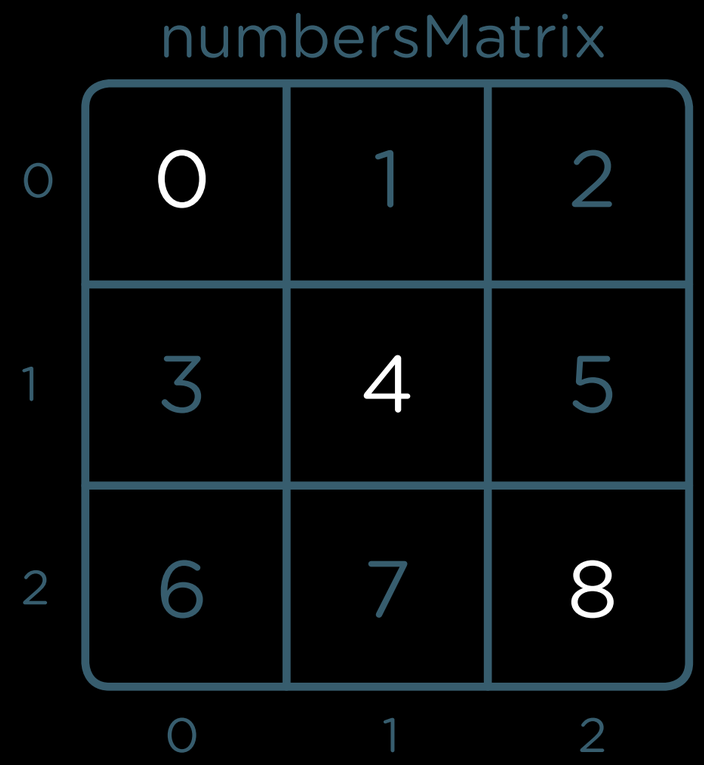



```
int[] numbers = { 0, 1, 2 };
// println(i);
```



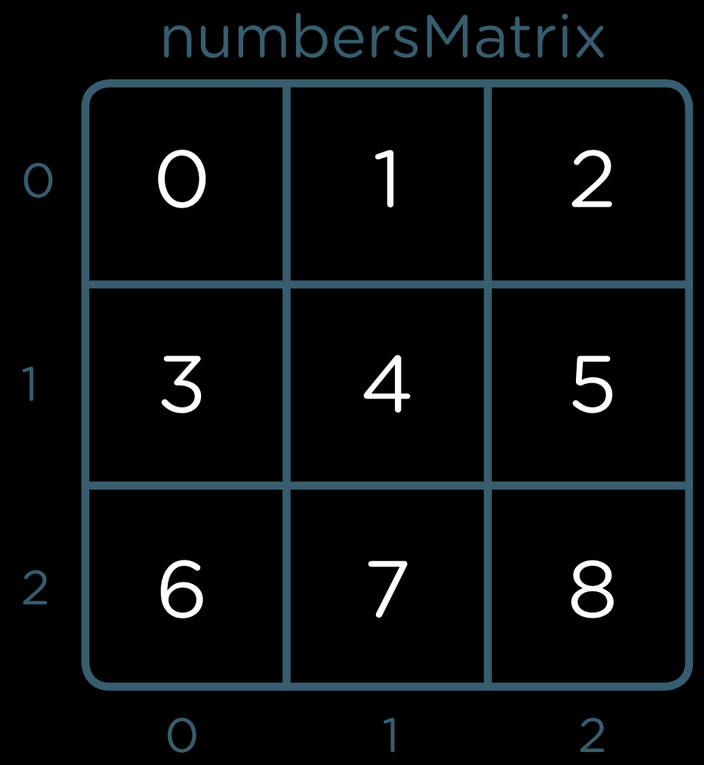

```
String[] people = new String[3];
people[0] = "Mark";
people[1] = "Annie";
people[2] = "David";
// println(people);
```

2d Arrays




```
int[][] numbersMatrix = {
  {0, 1, 2},
  {3, 4, 5},
  {6, 7, 8},
};

// println(numbersMatrix[0][0]);
// println(numbersMatrix[1][1]);
// println(numbersMatrix[2][2]);
```

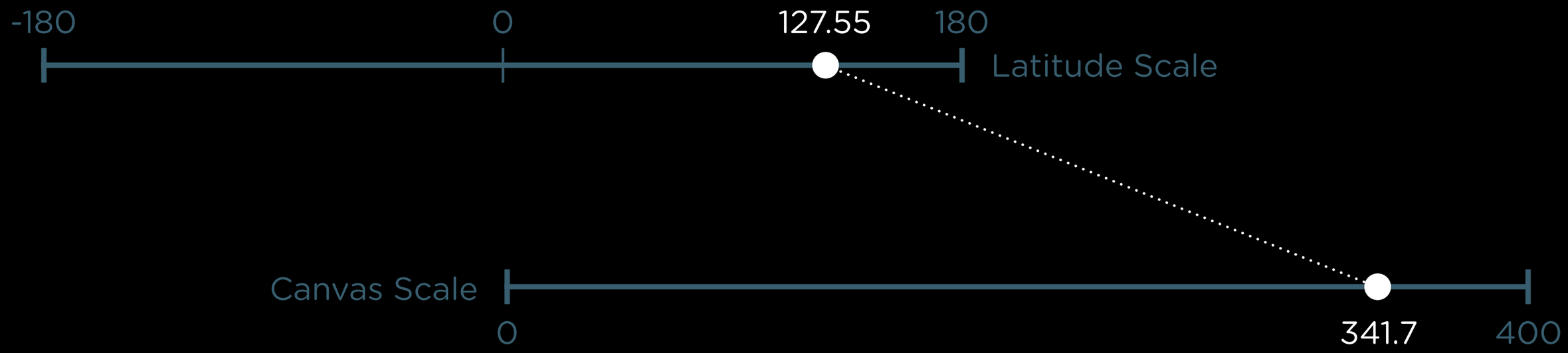
for() Loops




```
for(int i=0; i<3; i++){
// println(numbersMatrix[i][0]);
// println(numbersMatrix[i][1]);
// println(numbersMatrix[i][2]);
};
```

map() Function

```
map(127.55, -180, 180, 0, width);
```



Processing Reference

STRUCTURE
()
,
.
/* */
/** */
//
;
=
[]
{}
catch
class
draw()
exit()
extends
false
final
implements
import
loop()
new
noLoop()
null
popStyle()
private
public
pushStyle()
redraw()
return
setup()
static
super
this
true
try
void

ENVIRONMENT
cursor()
displayHeight
displayWidth
focused
frameCount
frameRate()
frameRate
height
noCursor()
size()
width

DATA
Primitive
boolean
byte
char
color
double
float
int
long

Composite
Array
FloatDict
FloatList
HashMap
IntDict
IntList
JSONArray
JSONObjec t
Object
String
StringDict
StringList
Table
TableRow
XML

Conversion
binary()
boolean()
byte()
char()
float()
hex()
int()
str()
unbinary()
unhex()

String Functions
join()
match()
matchAll()
nf()
nfc()
nfp()
nfs()
split()
splitTokens()
trim()

Array Functions
append()
arrayCopy()
concat()
expand()
reverse()
shorten()
sort()
splice()
subset()

CONTROL
Relational Operators
!=
<
<=
==
>
>=

Iteration
for
while

Conditionals
?:
break
case
continue
default
else
if
switch

Logical Operators
!
&&
||

SHAPE
createShape()
loadShape()
PShape

2D Primitives
arc()
ellipse()
line()
point()
quad()
rect()
triangle()

Curves
bezier()
bezierDetail()
bezierPoint()
bezierTangent()
curve()
curveDetail()
curvePoint()
curveTangent()
curveTightness()

3D Primitives
box()
sphere()
sphereDetail()

Attributes
ellipseMode()
noSmooth()
rectMode()
smooth()
strokeCap()
strokeJoin()
strokeWeight()

Vertex
beginContour()
beginShape()
bezierVertex()
curveVertex()
endContour()
endShape()
quadraticVertex()
vertex()

Loading & Displaying
shape()
shapeMode()

INPUT
Mouse
mouseButton
mouseClicked()
mouseDragged()
mouseMoved()
mousePressed()
mousePressed
mouseReleased()
mouseWheel()
mouseX
mouseY
pmouseX
pmouseY

Keyboard
key
keyCode
keyPressed()
keyPressed
keyReleased()
keyTyped()

Files
BufferedReader
createInput()
createReader()
loadBytes()
loadJSONArray()
loadJSONObject()
loadStrings()
loadTable()
loadXML()
open()
parseXML()
saveTable()
selectFolder()
selectInput()

Time & Date
day()
hour()
millis()
minute()
month()
second()
year()

OUTPUT
Text Area
print()
println()
Image
save()
saveFrame()

Files
beginRaw()
beginRecord()
createOutput()
createWriter()
endRaw()
endRecord()
PrintWriter
saveBytes()
saveJSONArray()
saveJSONObject()
saveStream()
saveStrings()
saveXML()
selectOutput()

TRANSFORM
applyMatrix()
popMatrix()
printMatrix()
pushMatrix()
resetMatrix()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
shearX()
shearY()
translate()

LIGHTS, CAMERA
Lights
ambientLight()
directionalLight()
lightFalloff()
lights()
lightSpecular()
noLights()
normal()
pointLight()
spotLight()

Camera
beginCamera()
camera()
endCamera()
frustum()
ortho()
perspective()
printCamera()
printProjection()

Coordinates
modelX()
modelY()
modelZ()
screenX()
screenY()
screenZ()

Material Properties
ambient()
emissive()
shininess()
specular()

COLOR
Setting
background()
clear()
colorMode()
fill()
noFill()
noStroke()
stroke()

Creating & Reading
alpha()
blue()
brightness()
color()
green()
hue()
lerpColor()
red()
saturation()

IMAGE
createImage()
PImage

Loading & Displaying
image()
imageMode()
loadImage()
noTint()
requestImage()
tint()

Textures
texture()
textureMode()
textureWrap()

Pixels
blend()
copy()
filter()
get()
loadPixels()
pixels[]
set()
updatePixels()

RENDERING
blendMode()
createGraphics()
PGraphics

Shaders
loadShader()
PShader
resetShader()
shader()

TYPOGRAPHY
PFont

Loading & Displaying
createFont()
loadFont()
text()
textFont()

Attributes
textAlign()
textLeading()
textMode()
textSize()
textWidth()

Metrics
textAscent()
textDescent()

MATH
PVector

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

Bitwise Operators
&
<<
>>
|

Calculation
abs()
ceil()
constrain()
dist()
exp()
floor()
lerp()
log()
mag()
map()
max()
min()
norm()
pow()
round()
sq()
sqrt()

Trigonometry
acos()
asin()
atan()
atan2()
cos()
degrees()
radians()
sin()
tan()

Random
noise()
noiseDetail()
noiseSeed()
random()
randomGaussian()
randomSeed()

CONSTANTS
HALF_PI
PI
QUARTER_PI
TAU
TWO_PI

Data Formats

CSV

```
Item, Price, Amount
Apple, $0.50, 3
Milk, $3.00, 1
Chips, $1.00, 1
```

Comma Separated Values

XML

```
<?xml version="1.0" encod-
ing="UTF-8"?>
<Inventory>
  <Item>
    <Name>Apple</Name>
    <Price> $0.50</Price>
    <Amount>3</Amount>
  </Item>
  <Item>
    <Name>Milk</Name>
    <Price> $3.00</Price>
    <Amount>1</Amount>
  </Item>
  <Item>
    <Name>Chips</Name>
    <Price> $1.00</Price>
    <Amount>1</Amount>
  </Item>
</Inventory>
```

Extensible Markup Language

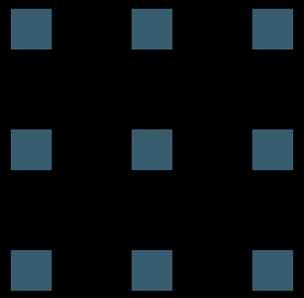
JSON

```
[
   {
      “Name”: “Apple”,
      “Price”: “$0.50”,
      “Amount”: “3”
   },
   {
      “Name”: “Milk”,
      “Price”: “$3.00”,
      “Amount”: “1”
   },
   {
      “Name”: “Chips”,
      “Price”: “$1.00”,
      “Amount”: “1”
   }
]
```

JavaScript Object Notation

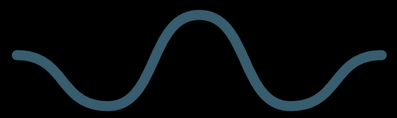
Drawing Structures

Rows

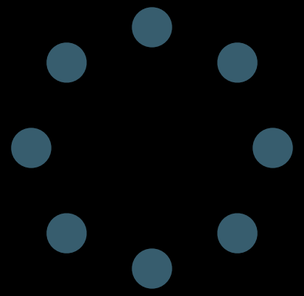


Grid

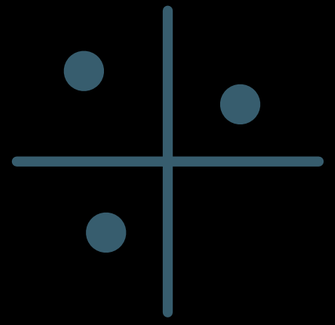
Concentric



Lines



Radial



Coordinates

Writing Better Code

“90% of coding is debugging.
The other 10% is writing bugs”

Bram Cohen, BitTorrent

Writing Better Code

“Debugging is like being the detective
in a crime movie
where you are also the murderer.”

Filipe Fortes, Flipboard

Writing Better Code

How to avoid code that is valid,
but doesn’t work the way you intended?

Variables: Constants

Fast but can lead to errors:

```
ellipse(posX, posY, 10*data[i], 10*data[i]);
```

Establish constants once:

```
int rectScale = 10;
rect(posX, posY, rectScale*data[i], rectScale*data[i]);
```

Variables: Naming

Fast but cryptic:

```
int s = 10;
int m = 50;
```

Use precise and concise names:

```
int globalScale = 10;
int margin = 50;
```

DRY: Don’t Repeat Yourself

Repetitive and fragile:

```
ellipse(posX, posY, 2*sqrt(area/PI), 2*sqrt(area/PI));
```

Easier to read, safer and less redundant:

```
float markerSize = 2*sqrt(area/PI);
ellipse(posX, posY, markerSize, markerSize);
```

DRY: Don’t Repeat Yourself

Create a function when code repeats frequently:

```
void draw(){
  float markerSize = circleDiameter(area);
}

float circleDiameter(float area) {
  area = 2*sqrt(area/PI);
  return area;
}
```

Neatness Counts: Proper Undentation

Unformatted code:

```
void draw(){
int x = 10;
for(int i=0; i<x; i++){
text(“hi”, x, height/2);
}
}
```

Formatted with Command-T:

```
void draw() {
  int x = 10;
  for (int i=0; i<x; i++) {
    text(“hi”, x, height/2);
  }
}
```

Neatness Counts: Remove Unused Elements

Unused elements:

```
int rectPosX = 10;
int rectPosY = 10;
int ellipsePosX = 20;
int ellipsePosY = 20;
int markerSize = 4;

ellipse(ellipsePosX,
ellipsePosY, markerSize,
markerSize);
```

Cleaned Code:

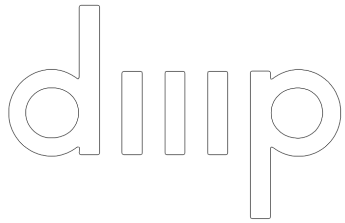
```
int posX = 10;
int posY = 10;
int markerSize = 4;

ellipse(posX, posY,
markerSize, markerSize);
```

Debugging

- ArrayIndexOutOfBoundsException
- println()
- Transparency
- stroke()
- Annotations (text or handles)
- Precision issues (float vs int)
- frameRate & frameCount

Refinements

- Shapes
- Custom font
- textMode(MODEL)
- Rounded rects
- HSB vs RGB color mode
- Save PNG / JPG / TGA / TIF
- Save PDF
- randomSeed()
- Conditional Labeling
- Formatting Numbers (rounding & commas)



KREATIVES PROGRAMMIEREN

AN INFOVIS TOOLKIT FOR PROCESSING