PAT 204/504 (Fall 2024)

Creative Coding

Lecture 5: Data Types & Arrays

Instructor: Hao-Wen Dong

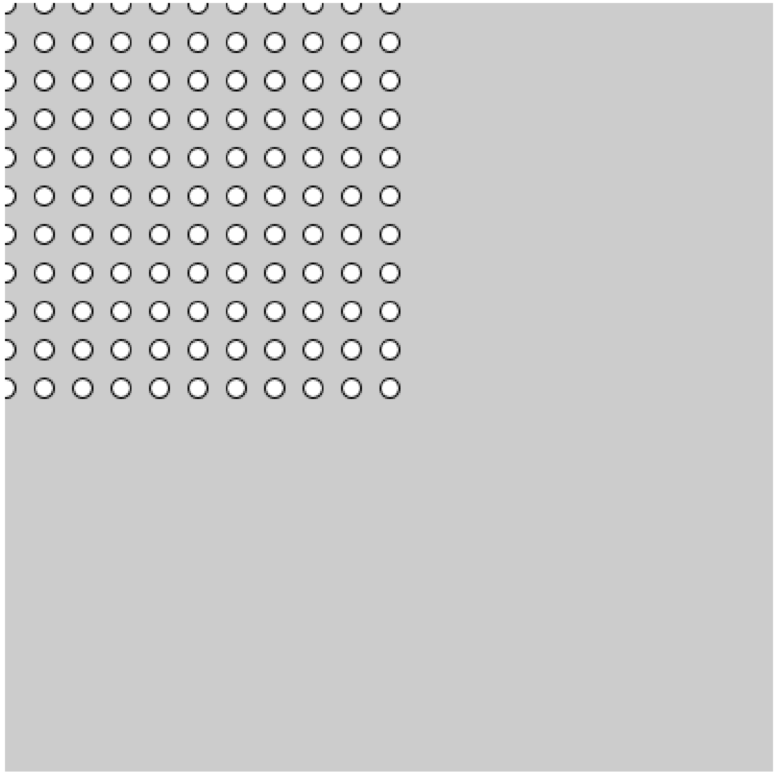
SCHOOL OF MUSIC, THEATRE & DANCE
PERFORMING ARTS TECHNOLOGY
UNIVERSITY OF MICHIGAN

(Recap) **Exercise**: Grid of Circles

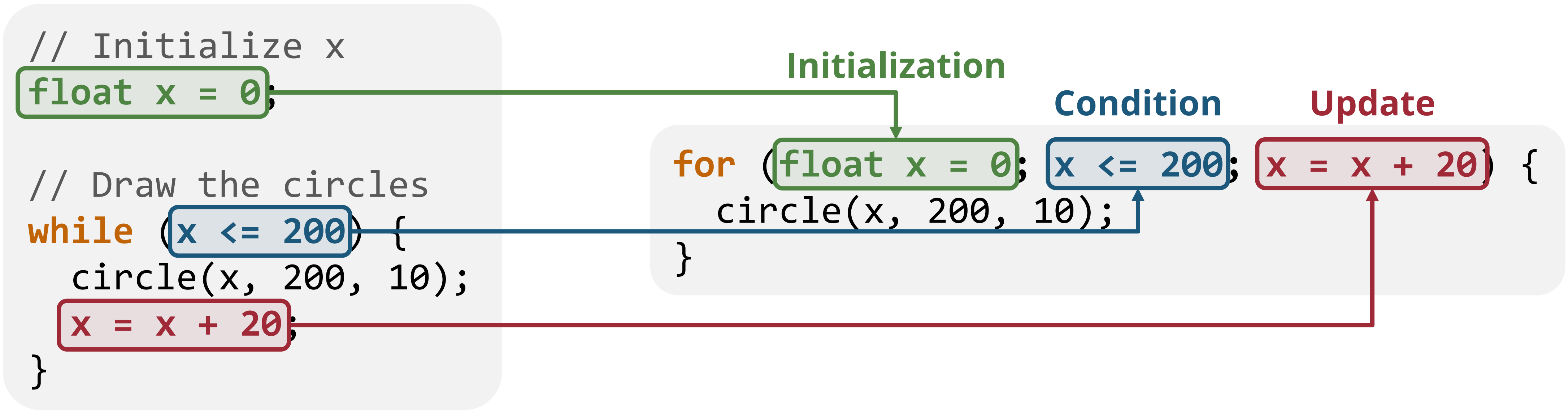
- Draw a grid of circles using `while` loops
- You'll need two `while` loops
  - One inside the other, i.e., ***nested* loops**

```
float x = 0, y = 0;

while (x <= 200) {
  y = 0;
  while (y <= 200) {
    circle(x, y, 10);
    y += 20;
  }
  x += 20;
}
```

Don't forget to reset y to 0



for Loop



```
// Initialize x
float x = 0;

// Draw the circles
while (x <= 200) {
  circle(x, 200, 10);
  x = x + 20;
}
```

Initialization

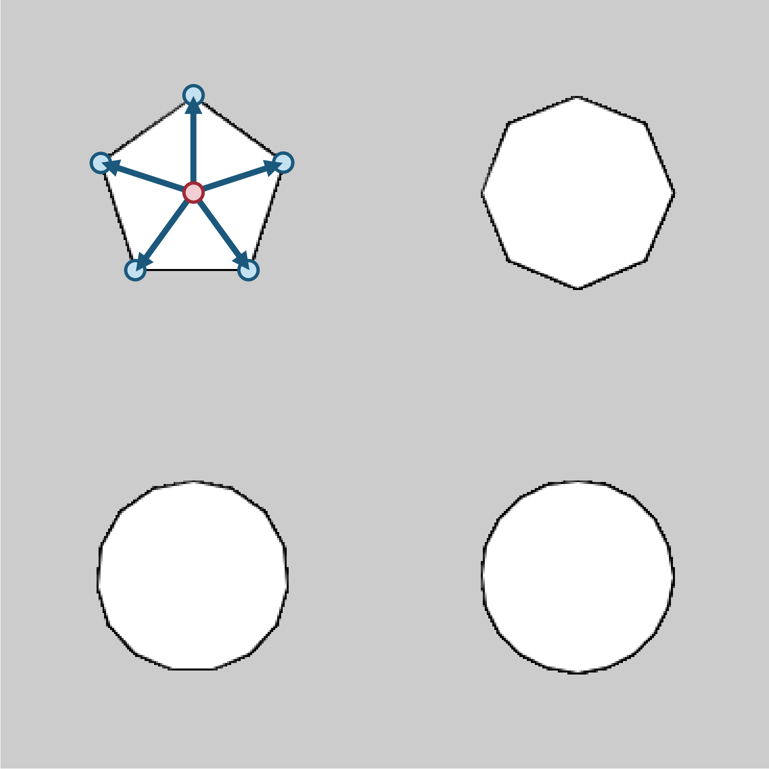
Condition

Update

```
for (float x = 0; x <= 200; x = x + 20) {
  circle(x, 200, 10);
}
```

(Recap) **Exercise**: Regular Polygons

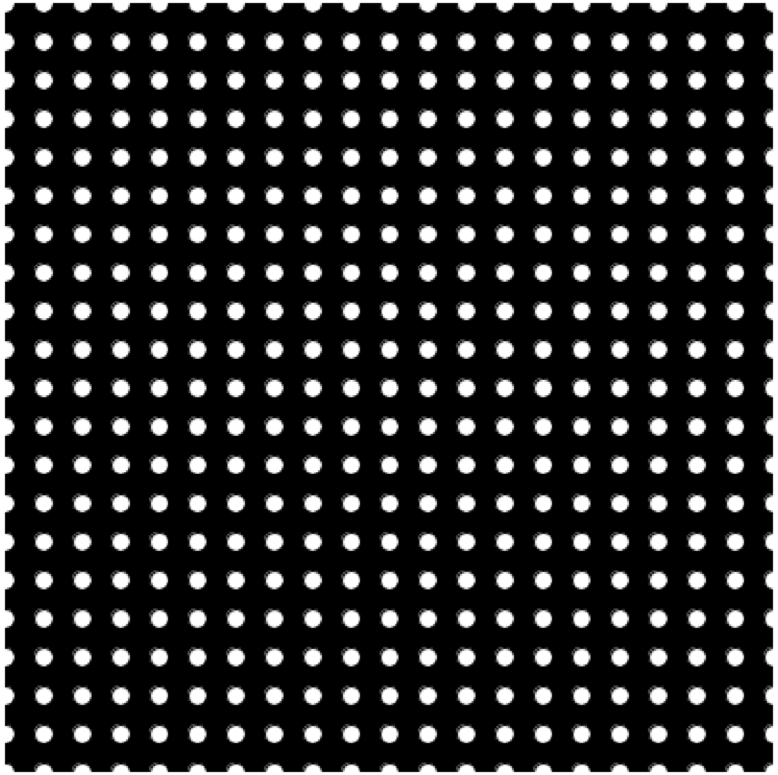
```
void polygon(float x, float y, float radius, int n) {
  float vertexX, vertexY;
  beginShape();
  for (float a = 0; a < TWO_PI; a += TWO_PI / n) {
    vertexX = x + radius * cos(a - HALF_PI);
    vertexY = y + radius * sin(a - HALF_PI);
    vertex(vertexX, vertexY);
  }
  endShape(CLOSE);
}
```



(Recap) **Exercise**: Grid of Circles

- Draw a grid of circles using for loops
- Again, you'll need two nested for loops

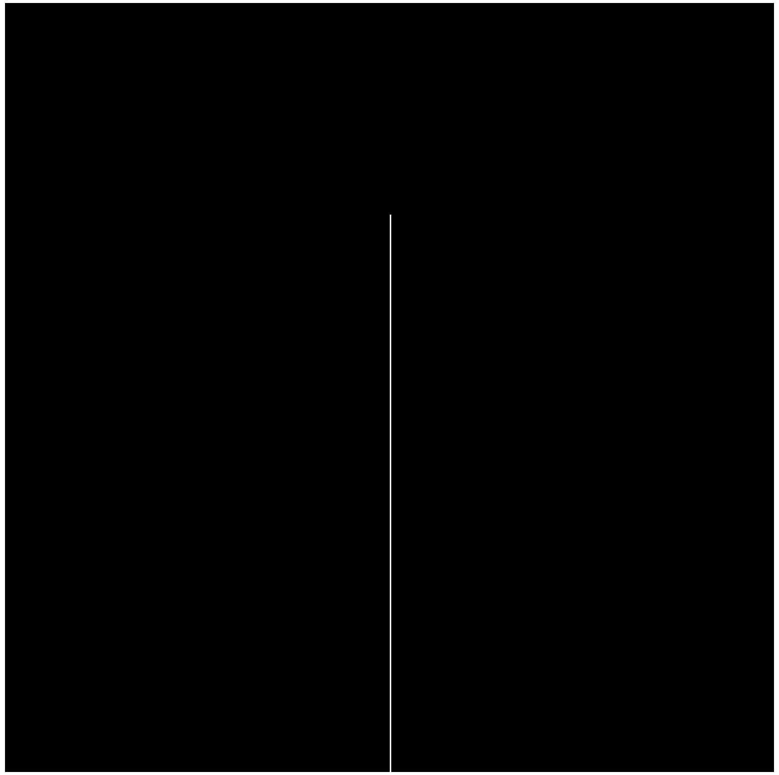
```
for (int i = 0; i <= 20; i++) {
  for (int j = 0; j <= 20; j++) {
    circle(i * 20, j * 20, 10);
  }
}
```



(Recap) Example: Recursive Tree

```
void branch(float h) {
  if (h < 2) break;

  // Right branch
  pushMatrix();
  rotate(theta);
  line(0, 0, 0, -h * scale);
  translate(0, -h * scale);
  branch(h * scale);
  popMatrix();

  // Left branch
  pushMatrix();
  rotate(-theta);
  line(0, 0, 0, -h * scale);
  translate(0, -h * scale);
  branch(h * scale);
  popMatrix();
}
```

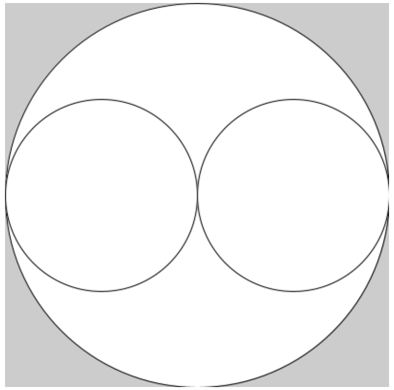
Stop condition



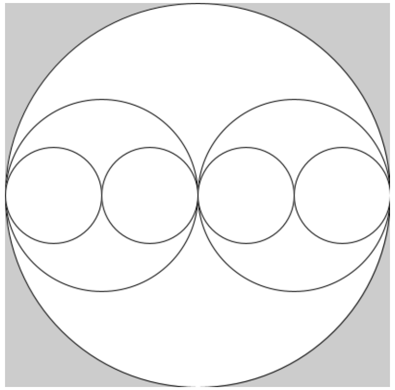
Exercise: Recursive Circles

- Recursively calling a function

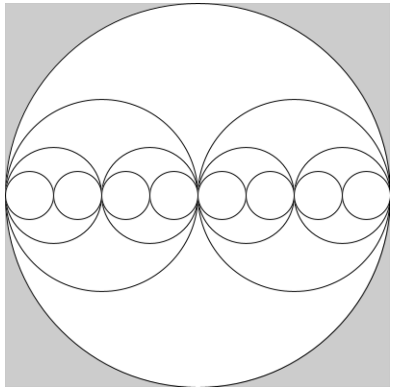
Level = 1



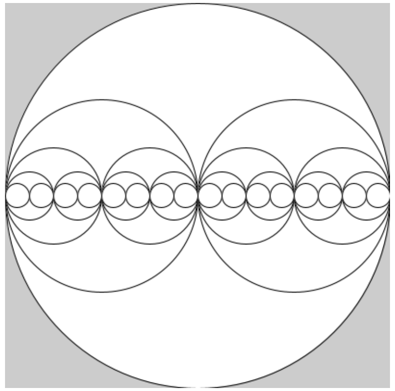
Level = 2



Level = 3

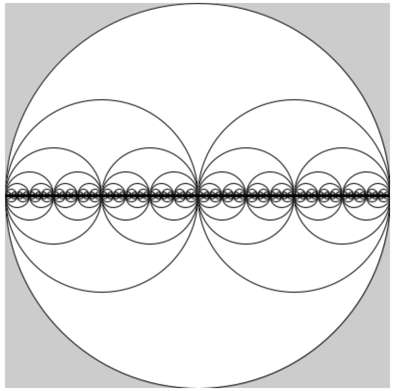


Level = 4



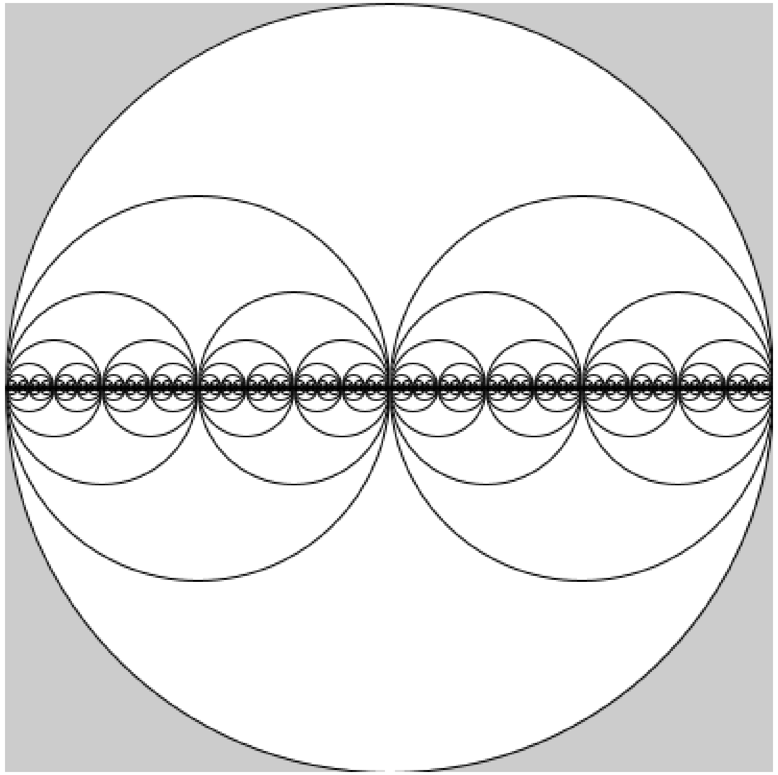
…

Level → ∞



Exercise: Recursive Circles

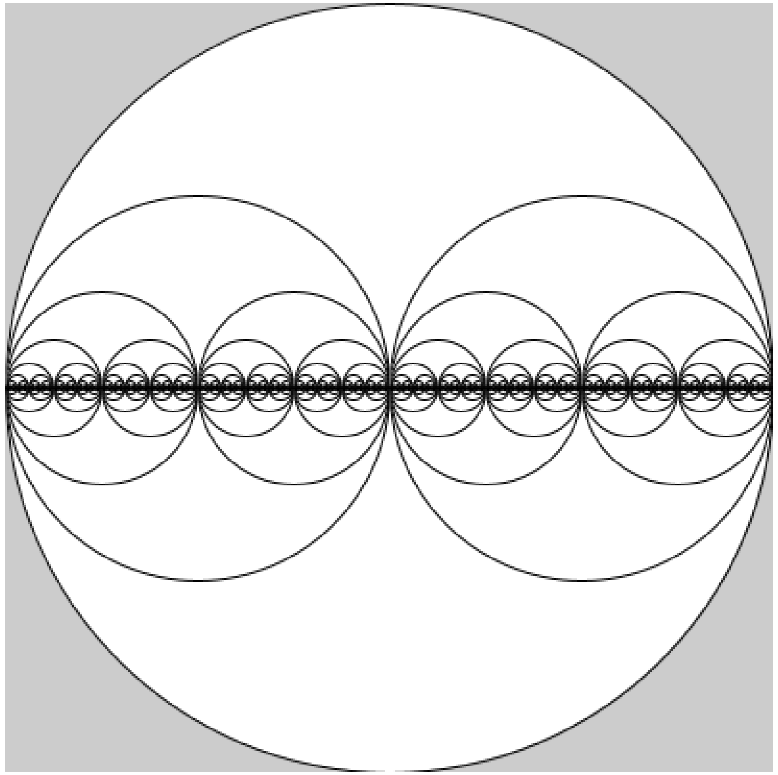
```
void drawCircles(float x, float y, float w) {

  // YOUR CODE HERE

}

void draw() {
  circle(200, 200, 400);
  drawCircles(200, 200, w);
}
```



Exercise: Recursive Circles

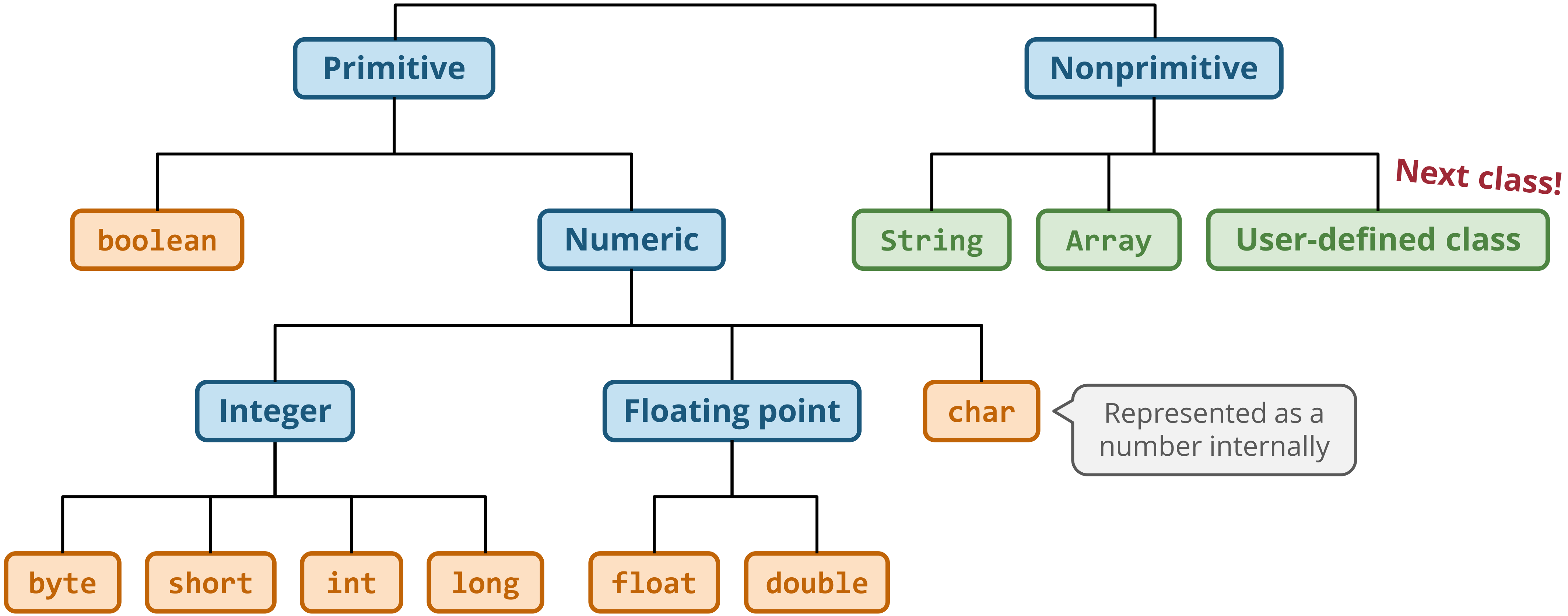
```
void drawCircles(float x, float y, float w) {
  if (w < 1) return;
  circle(x - w / 4, y, w / 2);
  drawCircles(x - w / 4, y, w / 2);

  circle(x + w / 4, y, w / 2);
  drawCircles(x + w / 4, y, w / 2);
}

void draw() {
  circle(200, 200, 400);
  drawCircles(200, 200, w);
}
```

Stop condition

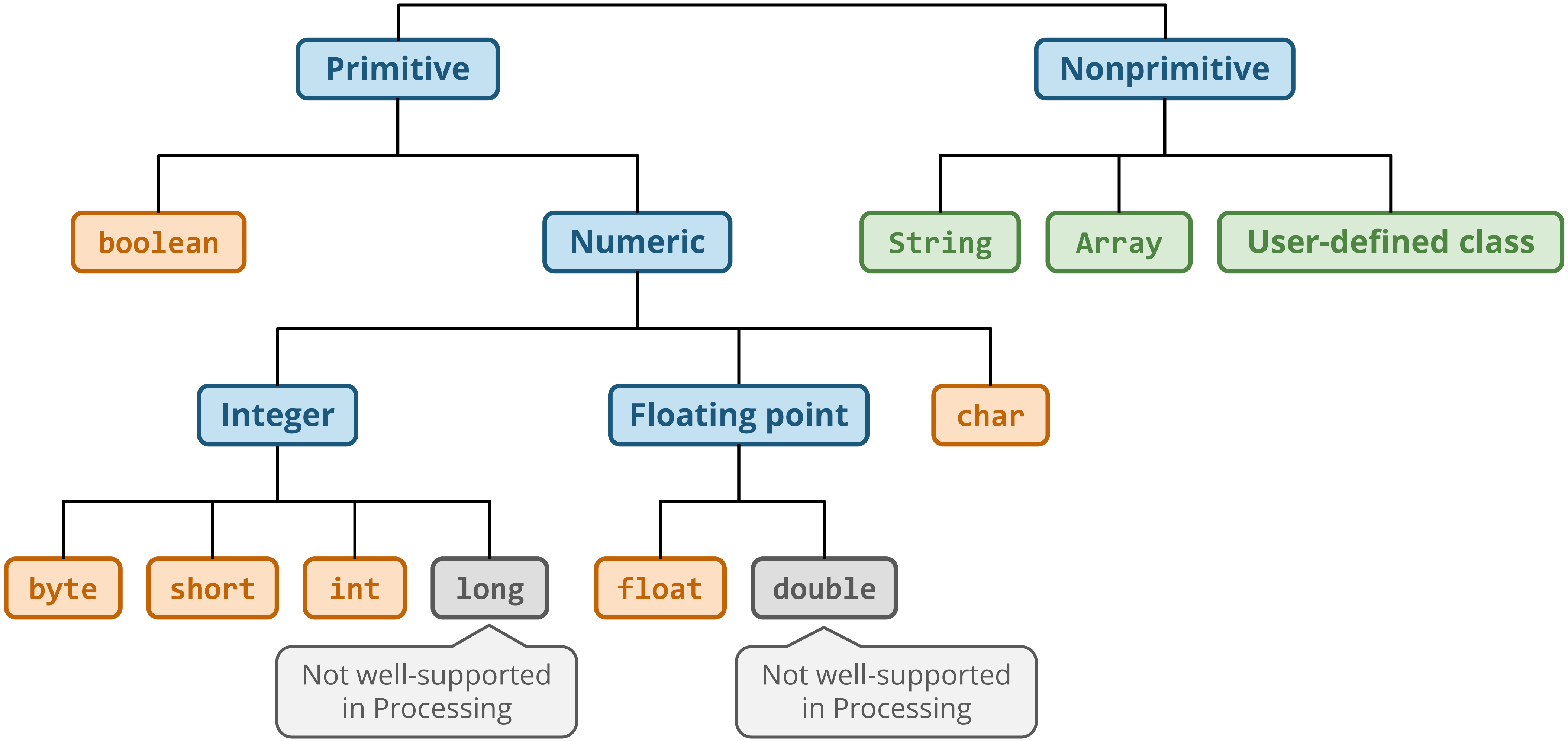


Data Types

Data Types


Primitive
Nonprimitive
boolean
Numeric
String
Array
User-defined class
Next class!
Integer
Floating point
char
Represented as a number internally
byte
short
int
long
float
double

Data Types


Primitive
Nonprimitive
boolean
Numeric
String
Array
User-defined class
Integer
Floating point
char
byte
short
int
long
float
double
Not well-supported in Processing
Not well-supported in Processing

Primitive Data Types

	Range	Default	Bytes	
boolean	true, false	false	1	**Why?**
byte	-128 ~ 127	0	1	
int	$-2^{31} \sim 2^{31}-1$	0	4	
long	$-2^{63} \sim 2^{63}-1$	0	8	
float	±1.4 E-45 ~ ±3.4 E38, ±∞, nan	0.0	4	
double	±4.9 E-324 ~ ±1.8 E308, ±∞, nan	0.0	8	
color	#00000000 ~ #FFFFFFFF	#00000000 (black)	4	**Why?**
char	0 to 65535 (letters, numbers, symbols, etc.)	‘\u0000’ (null character)	2	

Single precision

Double precision

Float.POSITIVE_INFINITY
Float.NEGATIVE_INFINITY

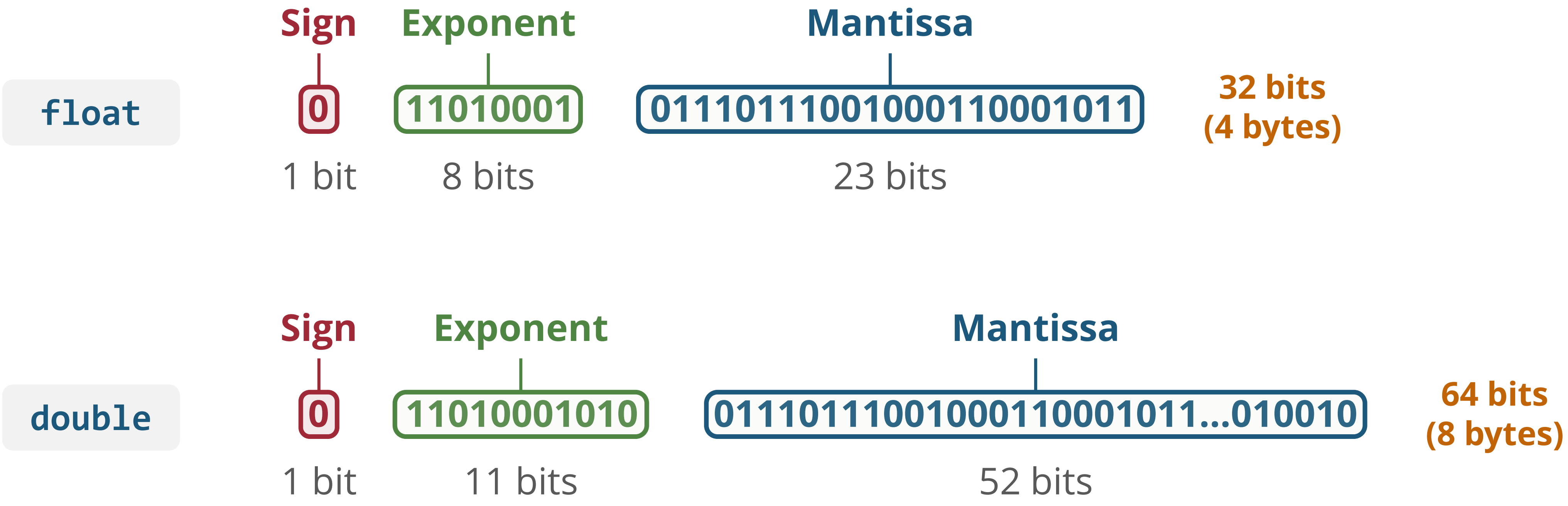
Float.NaN

‘0’, ‘a’, ‘ü’, ‘!’, ‘\n’

Primitive Data Types

	Range	Default	Bytes
boolean	true, false	false	1
byte	-128 ~ 127	0	1
int	-2^{31} ~ 2^{31}-1	0	4
long	-2^{63} ~ 2^{63}-1	0	8
float	±1.4 E-45 ~ ±3.4 E38, ±∞, nan	0.0	4
double	±4.9 E-324 ~ ±1.8 E308, ±∞, nan	0.0	8
color	#00000000 ~ #FFFFFFFF	#00000000 (black)	4
char	0 to 65535 (letters, numbers, symbols, etc.)	'\u0000' (null character)	2

Floating Numbers: `float` & `double`



`float`

Sign: 0 — 1 bit

Exponent: 11010001 — 8 bits

Mantissa: 01110111001000110001011 — 23 bits

32 bits
(4 bytes)

`double`

Sign: 0 — 1 bit

Exponent: 11010001010 — 11 bits

Mantissa: 01110111001000110001011...010010 — 52 bits

64 bits
(8 bytes)

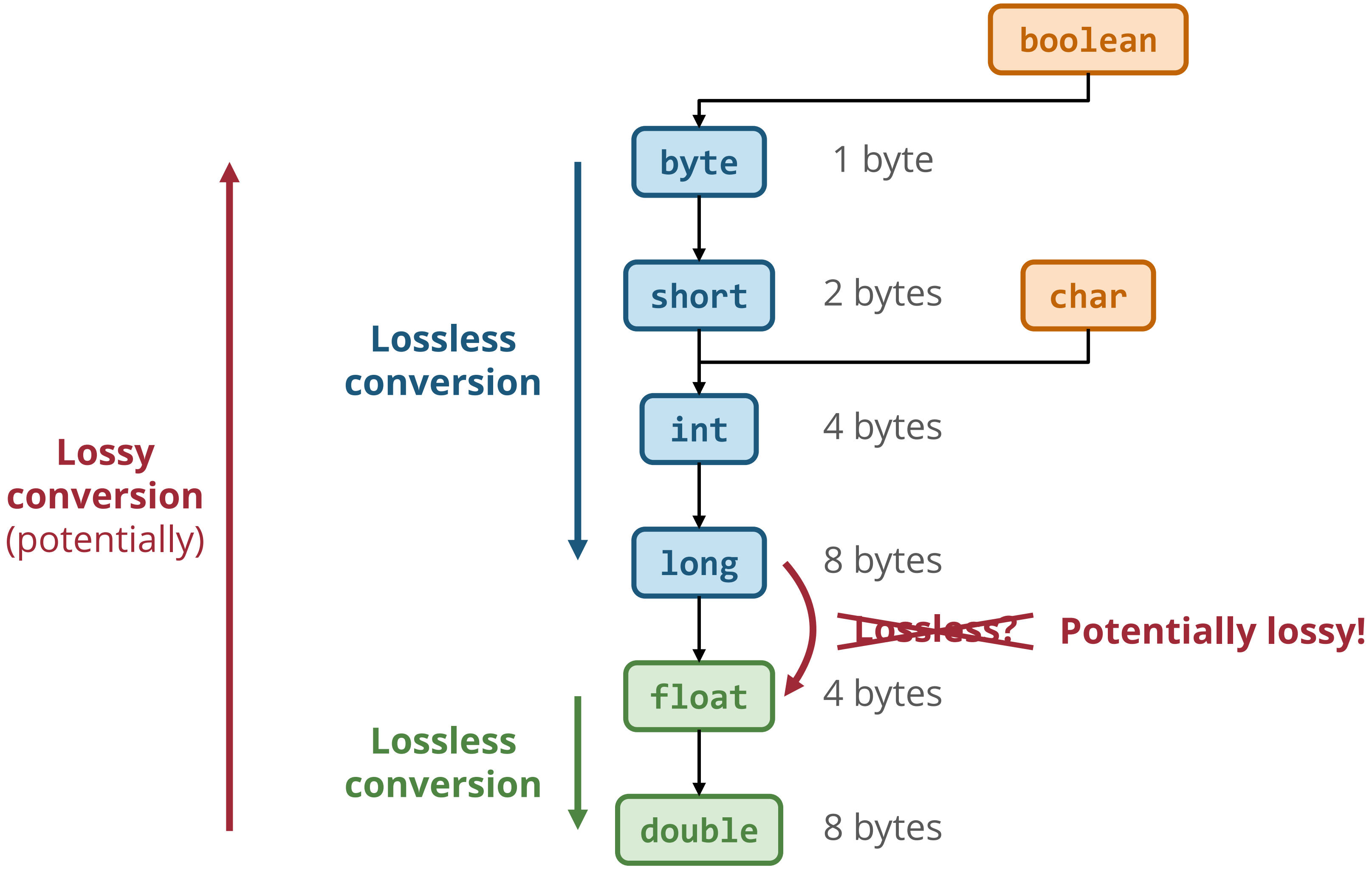
$$\text{sign} \times \text{mantissa} \times 2^{\text{exponent}}$$

Base 2!

Data Type Conversion

- Simply call the data type as a **function**
  - `boolean(x)`
  - `int(x)`
  - `float(x)`
  - `char(x)`
  - …

Data Type Conversion



Floating Point-to-Integer Conversion

- Behaves like rounding but not exactly
  - `int(1.2)` → 1
  - `int(1.8)` → 1
  - `int(-1.2)` → -1
  - `int(-1.8)` → -1
- For standard rounding behavior, use **`round()`**
  - `round(1.2)` → 1
  - `round(1.8)` → 2
  - `round(-1.2)` → -1
  - `round(-1.8)` → -2

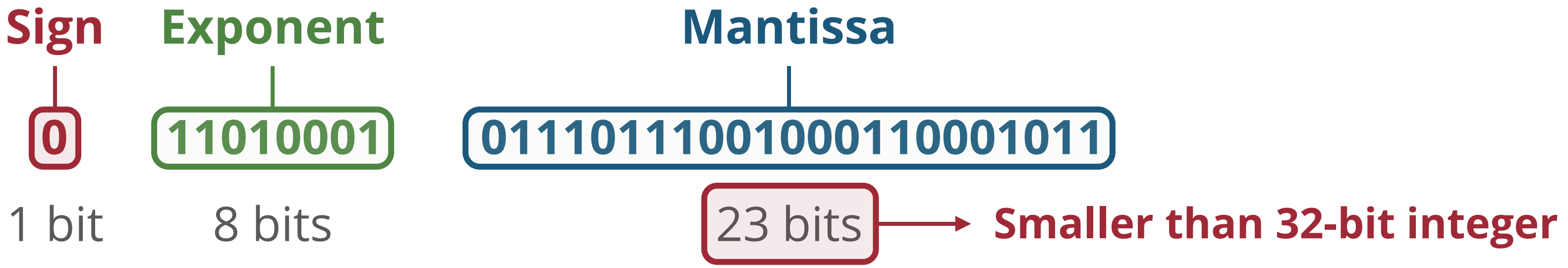
`round(1.5)` → 2
`round(-1.5)` → -1

Round up when in the middle

Lossy Conversion

```
int n = 1845654513;
println(n);          → 1845654513

float x = float(n);
println(x);          → 1.84565453E9

int m = int(x);
println(m);          → 1845654528
```

What happened?



Sign

0

1 bit

Exponent

11010001

8 bits

Mantissa

01110111001000110001011

23 bits → Smaller than 32-bit integer

Data Type Conversion – Another Approach

-
  ```
  int a = 1;
  float b = (float) a;
  ```

-
  ```
  float a = 1.5;
  int b = (int) a;
  ```

Implicit vs Explicit Type Conversion

Implicit

```
int a = 1;
float b = a;
print(b);
```

```
~~float a = 1.5;~~
~~int b = a;~~
~~print(b);~~
```

Explicit

```
int a = 1;
float b = float(a);
print(b);
```

```
float a = 1.5;
int b = int(a);
print(b);
```

```
int a = 1;
float b = (float) a;
print(b);
```

```
float a = 1.5;
int b = (int) a;
print(b);
```

Integer-to-Boolean Conversion

- **0** → **false**
- Everything else → **true**

Examples: Integer-to-Boolean Conversion

- `boolean(1)` **true**
- `boolean(0)` **false**
- `boolean(-1)` **true**
- `boolean(1.1)` **Doesn’t work**

Integer-to-Boolean Conversion

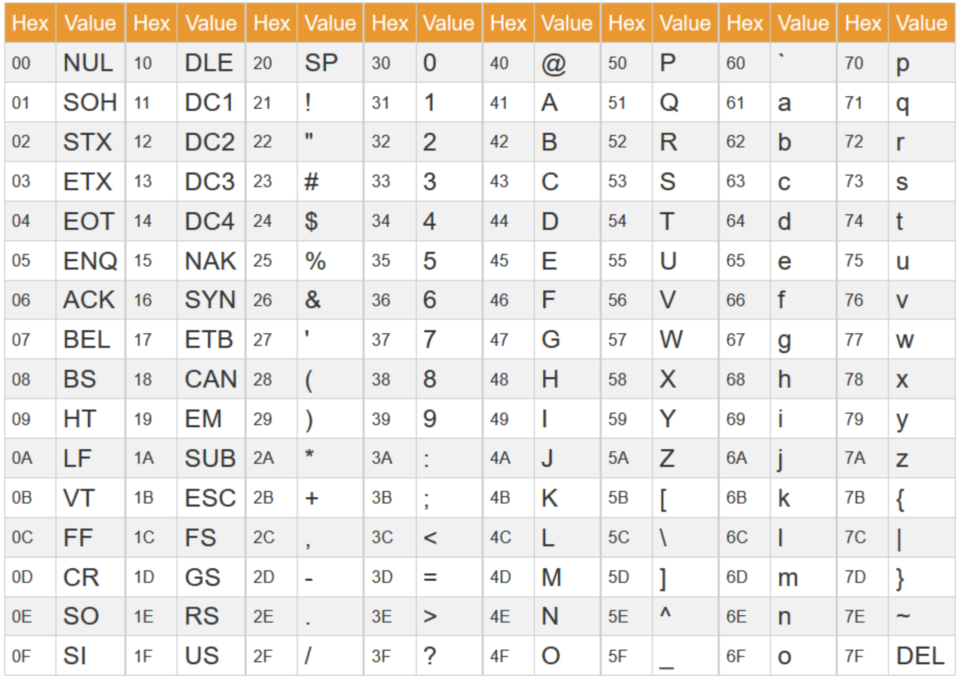
- “`true`” → **true**
- Everything else → **false**

Example: String-to-Boolean Conversion

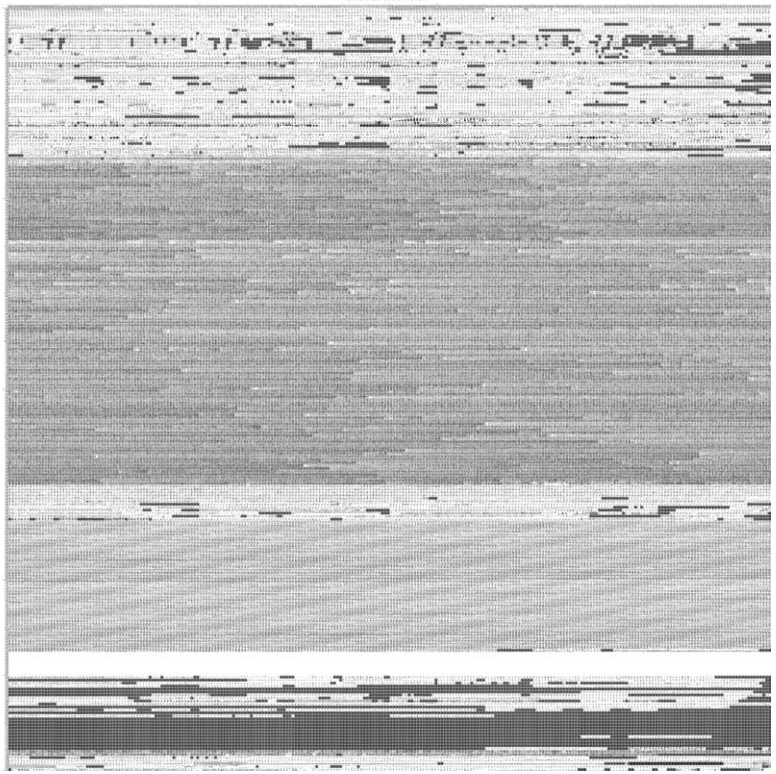
- boolean(“true”) true
- boolean(“false”) false
- boolean(“yes”) false
- boolean(“1”) false

Character Encoding in Processing: UTF-16

- Characters are represented as **numbers** in computers
- The default encoding is **UTF-16**



Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value	Hex	Value
00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	,	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL



Full character set of UTF-16

adriennedomingus.medium.com/character-encoding-and-the-internet-f452d37f2815

Examples: char-to-int & int-to-char Conversion

- char(65) → ‘A’
- int(‘A’) → 65

- char(97) → ‘a’
- int(‘a’) → 97

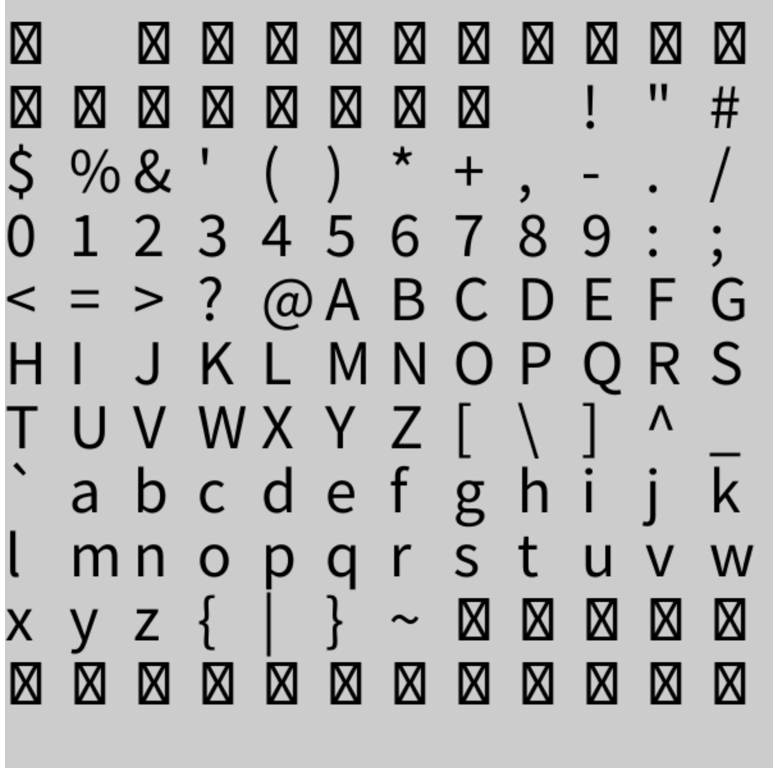
- char(192) → ‘À’
- char(224) → ‘à’

Exercise: Character Wall

- Print a matrix of characters using `for` loops

- **Approach 1**
  - Use a **nested `for` loop**
  - Loop over the **x index** and **y index**

- **Approach 2**
  - Use a **single `for` loop**
  - Loop over the **character code**



```
☒   ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒
☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒   ! " #
$ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ;
< = > ? @ A B C D E F G
H I J K L M N O P Q R S
T U V W X Y Z [ \ ] ^ _
` a b c d e f g h i j k
l m n o p q r s t u v w
x y z { | } ~ ☒ ☒ ☒ ☒ ☒
☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒ ☒
```

Two Ways of Looping

- **Approach 1**
  - Use a **nested `for` loop**
  - Loop over the **x index** and **y index**

```
for (int i = 0; i < 12; i++) {
  for (int j = 0; j < 12; j++) {
   char code = char(i + 12 * j);
   text(code, i * 50, j * 50);
  }
 }
```

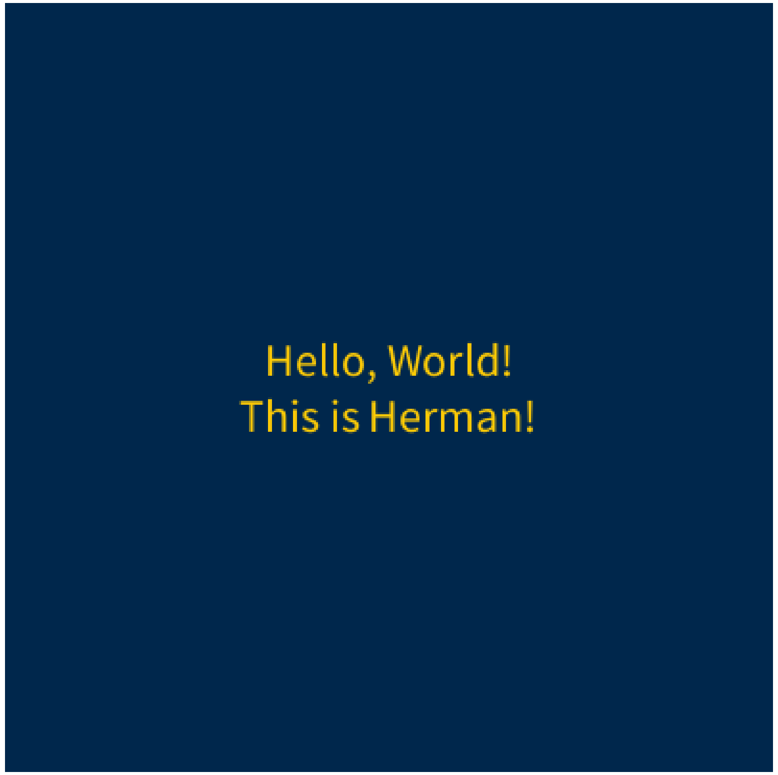
- **Approach 2**
  - Use a **single `for` loop**
  - Loop over the **character code**

```
for (char code = 0; code < 144; code++){
  idx = int(code);
  i = idx % 12;
  j = idx / 12;
  text(code, i * 50, j * 50);
 }
```

Common way to turn a 1D sequence into a 2D matrix

Text Rendering

Homework 1: Bouncing Hello World

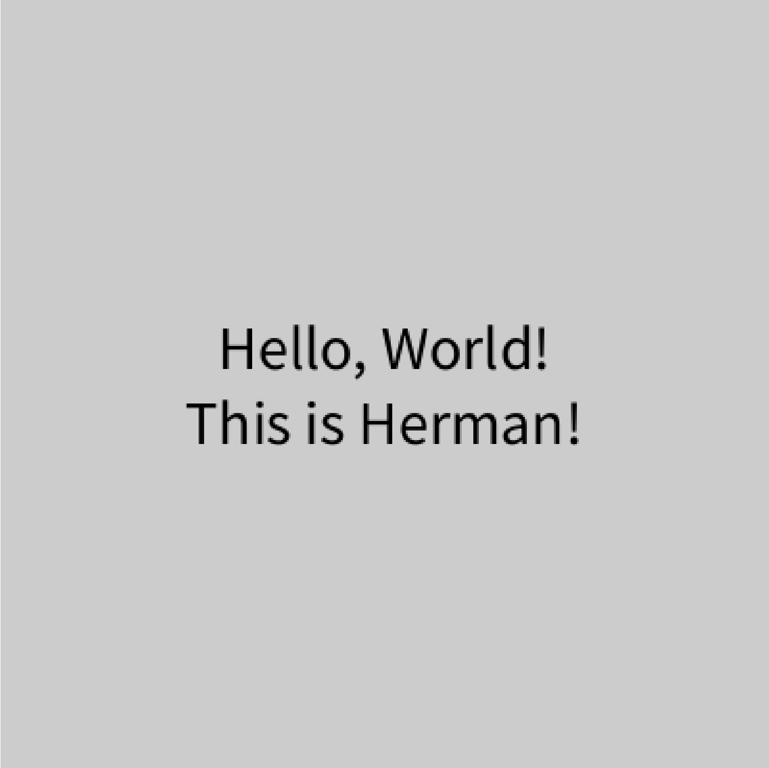


Hello, World!
This is Herman!

text()

- **text**(str, x, y)

```
String s = "Hello, World!\nThis is Herman!";
fill(0);
textSize(32)
textAlign(CENTER, CENTER);
text(s, 200, 200);
```

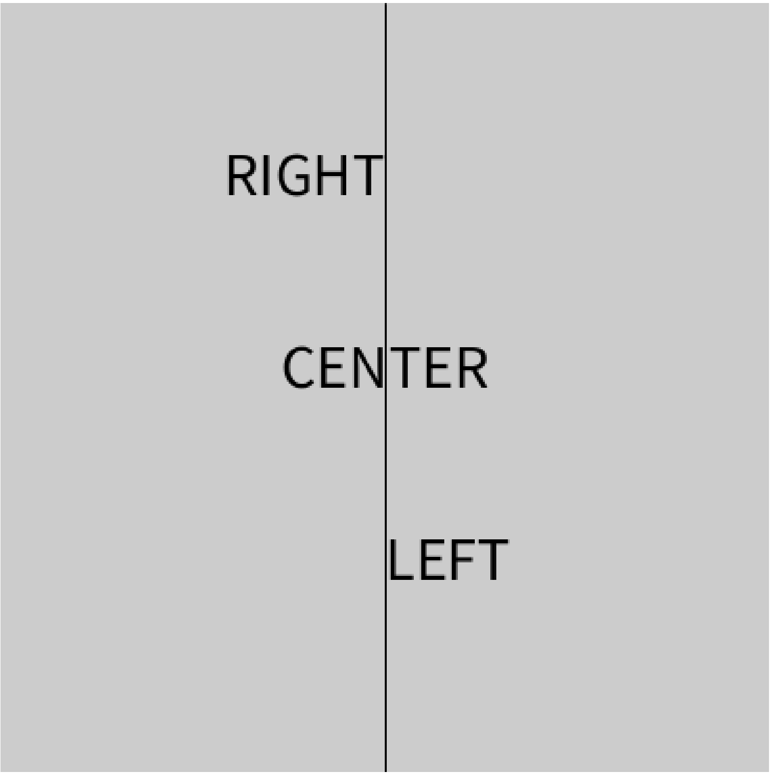


Hello, World!
This is Herman!

Text Alignment – textAlign()

- Set the **text alignment** and the **anchor point**

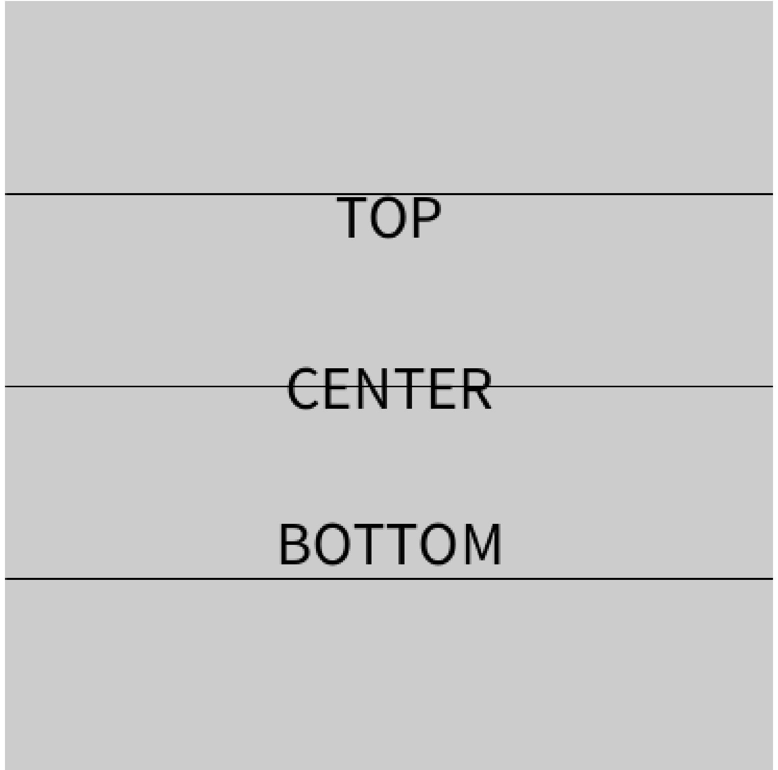
```
size(400, 400);
fill(0);
textSize(32);
line(200, 0, 200, 400);

textAlign(RIGHT);
text("RIGHT", 200, 100);

textAlign(CENTER);
text("CENTER", 200, 200);

textAlign(LEFT);
text("LEFT", 200, 300);
```



Text Alignment – textAlign()

- Set the **text alignment** and the **anchor point**

```
size(400, 400);
fill(0);
textSize(32);

textAlign(CENTER, TOP);
text("TOP", 200, 100);
line(0, 100, 400, 100);

textAlign(CENTER, CENTER);
text("CENTER", 200, 200);
line(0, 200, 400, 200);

textAlign(CENTER, BOTTOM);
text("BOTTOM", 200, 300);
line(0, 300, 400, 300);
```



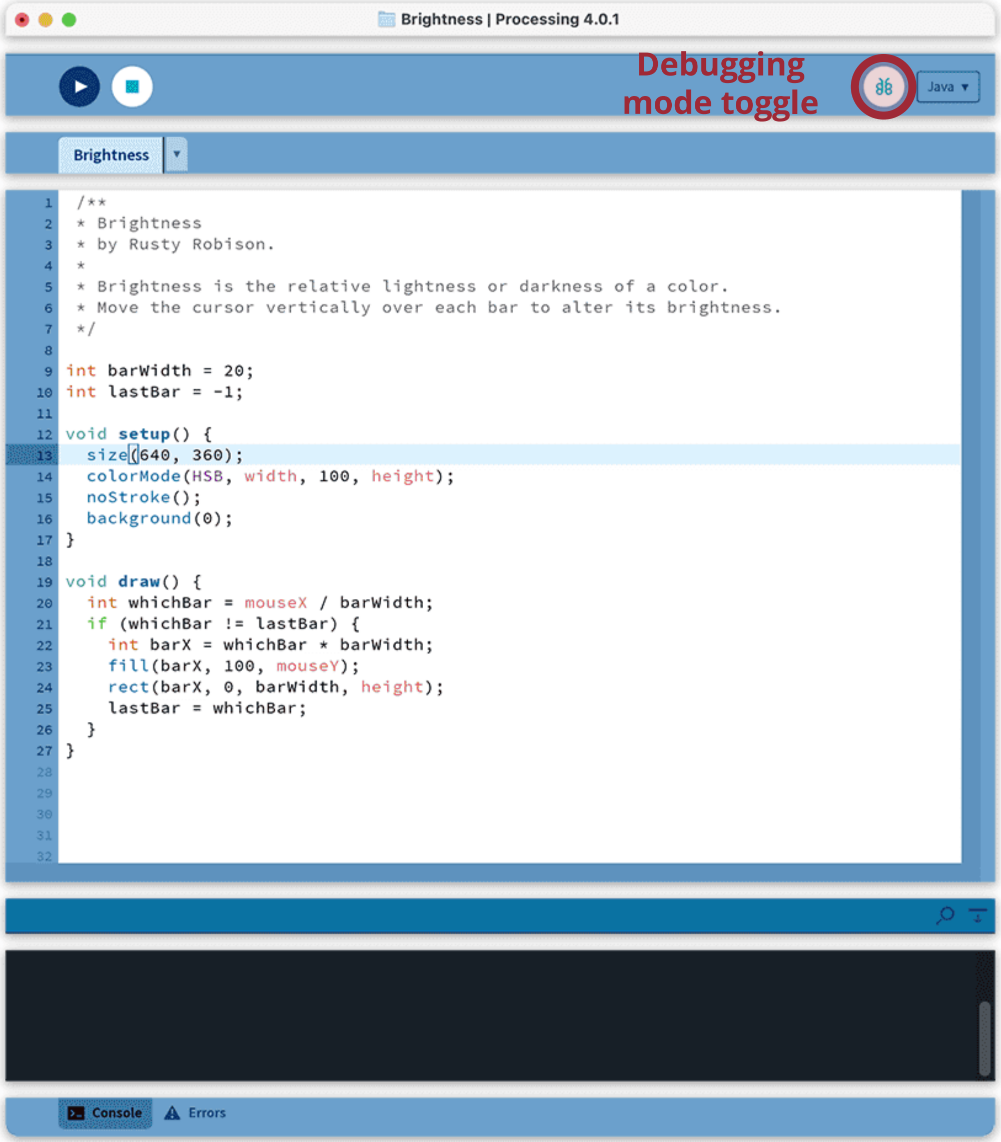
Debugging

`print()` & `println()`

- Output will show in the console at the bottom of the IDE
- Useful for debugging
- **`print()`** does not add a new line
- **`println()`** adds a new line
- New line is encoded as a special character **`‘\n’`**

Built-in Debugger

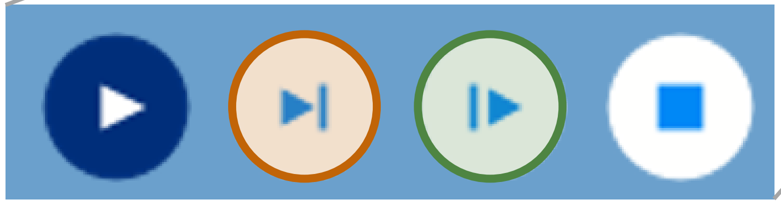
- Useful for **debugging**



Debugging mode toggle

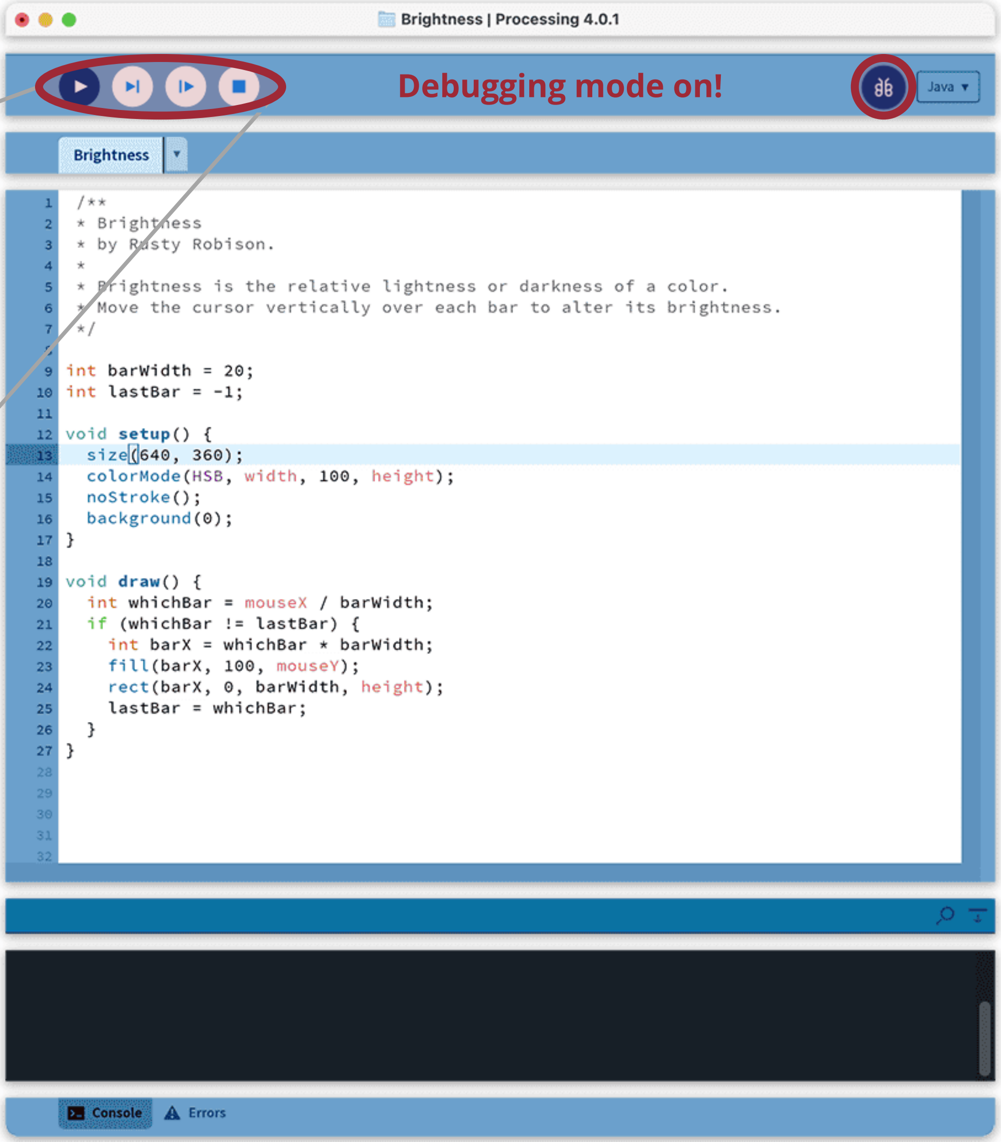
```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or darkness of a color.
 * Move the cursor vertically over each bar to alter its brightness.
 */

int barWidth = 20;
int lastBar = -1;

void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, height);
  noStroke();
  background(0);
}

void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

Built-in Debugger



Step

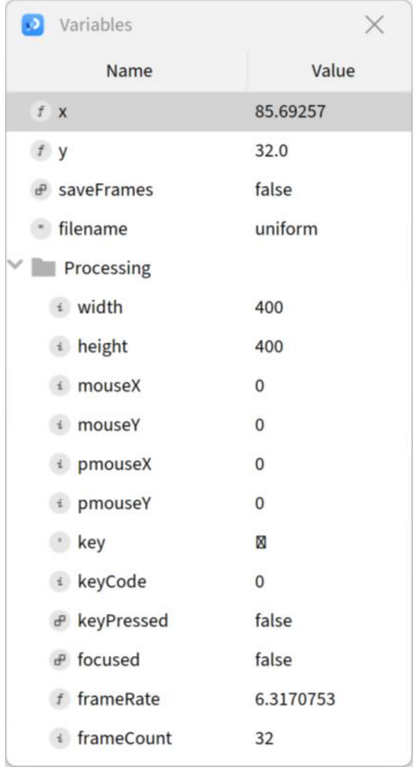
Continue
(until next break point)



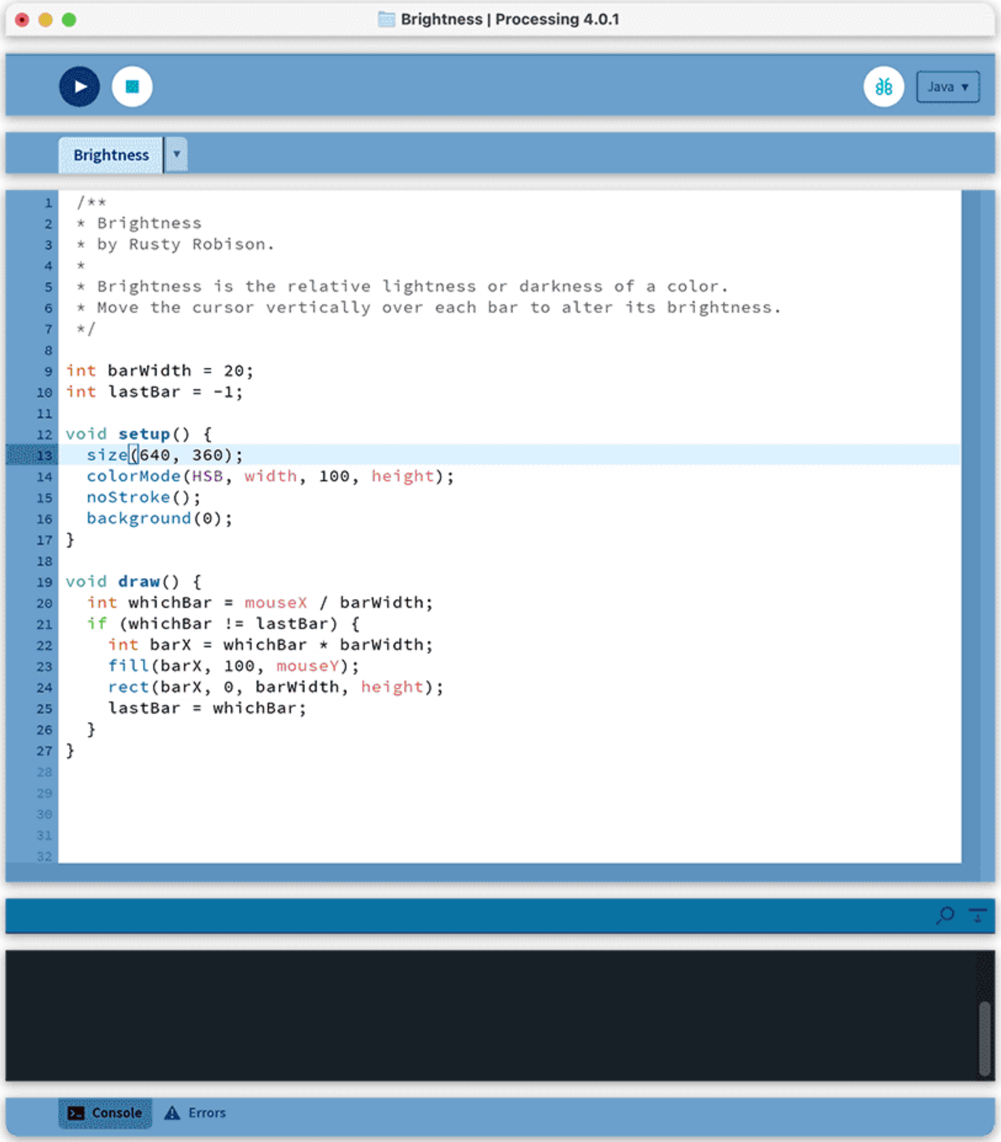
```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or darkness of a color.
 * Move the cursor vertically over each bar to alter its brightness.
 */

int barWidth = 20;
int lastBar = -1;

void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, height);
  noStroke();
  background(0);
}

void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

Variable Inspector

- Shows the variables and their values
  - Including built-in global variables



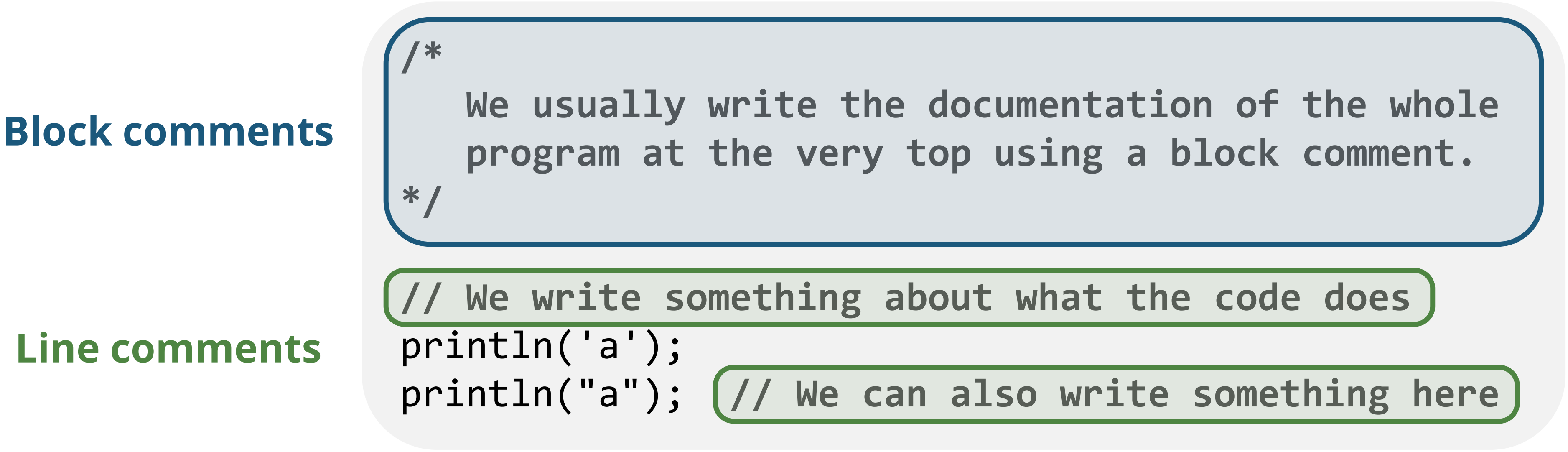
Name	Value
x	85.69257
y	32.0
saveFrames	false
filename	uniform
Processing	
width	400
height	400
mouseX	0
mouseY	0
pmouseX	0
pmouseY	0
key	⊠
keyCode	0
keyPressed	false
focused	false
frameRate	6.3170753
frameCount	32



```
/**
 * Brightness
 * by Rusty Robison.
 *
 * Brightness is the relative lightness or darkness of a color.
 * Move the cursor vertically over each bar to alter its brightness.
 */

int barWidth = 20;
int lastBar = -1;

void setup() {
  size(640, 360);
  colorMode(HSB, width, 100, height);
  noStroke();
  background(0);
}

void draw() {
  int whichBar = mouseX / barWidth;
  if (whichBar != lastBar) {
    int barX = whichBar * barWidth;
    fill(barX, 100, mouseY);
    rect(barX, 0, barWidth, height);
    lastBar = whichBar;
  }
}
```

Comments

- Line comments
- Block comments



Block comments

```
/*
   We usually write the documentation of the whole
   program at the very top using a block comment.
*/
```

Line comments

```
// We write something about what the code does
println('a');
println("a");  // We can also write something here
```

Arrays

Arrays

- Hold a fixed number of items of the **same data type**
  - Cannot change the length of an array once declared
  - Declare by adding **a pair of brackets** after the data type
- **`arr.length`** returns the length of the array
- **`arr[i]`** gives you the (i-1)-th item → **Index starts from 0!**

```
// Declare and initialize an array of ten integers
int x[] = new int[10];

x[0] = 1;
println(x[0]);  // 1

println(x.length)  // 10
```

Built-in Array Functions

- `arr.append()`
- `arr.expand()`
- `arr.shortern()`

- `concat(arr1, arr2)`
- `subset(arr)`
- `sort(arr)`
- `reverse(arr)`

All these functions return a new array!

String & char

- **String** acts like an **array of char**
- String supports all the built-in functions for arrays

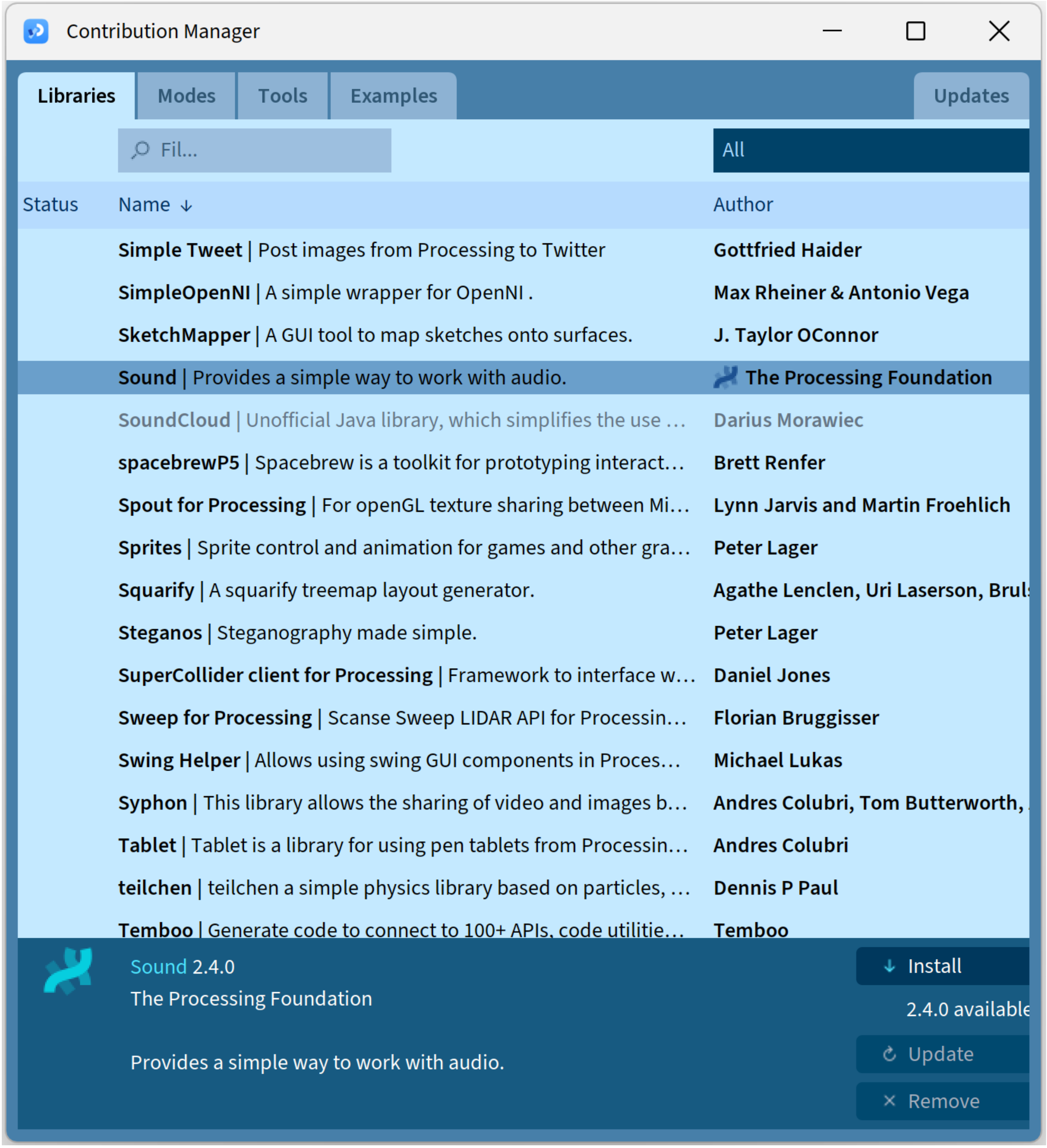
String & char – ‘a’ vs “a”

- **println(‘a’)** and **println(“a”)** both work!
- Single quote → char
- Double quote → String

Libraries

Library Manager

- Official Libraries maintained by the **Processing Foundation**
  - Sound
  - Video
  - Hardware I/O
  - JavaFX
- Many other libraries
  - Networking
  - GUI
  - Animation



Status	Name ↓	Author
	Simple Tweet | Post images from Processing to Twitter	Gottfried Haider
	SimpleOpenNI | A simple wrapper for OpenNI .	Max Rheiner & Antonio Vega
	SketchMapper | A GUI tool to map sketches onto surfaces.	J. Taylor OConnor
	Sound | Provides a simple way to work with audio.	The Processing Foundation
	SoundCloud | Unofficial Java library, which simplifies the use …	Darius Morawiec
	spacebrewP5 | Spacebrew is a toolkit for prototyping interact…	Brett Renfer
	Spout for Processing | For openGL texture sharing between Mi…	Lynn Jarvis and Martin Froehlich
	Sprites | Sprite control and animation for games and other gra…	Peter Lager
	Squarify | A squarify treemap layout generator.	Agathe Lenclen, Uri Laserson, Brul
	Steganos | Steganography made simple.	Peter Lager
	SuperCollider client for Processing | Framework to interface w…	Daniel Jones
	Sweep for Processing | Scanse Sweep LIDAR API for Processin…	Florian Bruggisser
	Swing Helper | Allows using swing GUI components in Proces…	Michael Lukas
	Syphon | This library allows the sharing of video and images b…	Andres Colubri, Tom Butterworth,
	Tablet | Tablet is a library for using pen tablets from Processin…	Andres Colubri
	teilchen | teilchen a simple physics library based on particles, …	Dennis P Paul
	Temboo | Generate code to connect to 100+ APIs, code utilitie…	Temboo

Sound 2.4.0
The Processing Foundation

Provides a simple way to work with audio.

Demo: Deep Vision Library

- Deep learning-powered computer vision library
- Support
  - Object detection
  - Object recognition
  - Object segmentation
  - Keypoint detection
  - Depth estimation
  - Style transfer
  - Superresolution

Homework 3: Spectrum Visualizer

- Modify the template code to implement a spectrum visualizer
- Instructions will be released on Gradescope
- Due at **11:59pm ET** on **September 20**
- Late submissions: **1 point deducted per day**

More on Colors

`color` is a Data Type

- You can define a variable of data type **`color`**
  - For example, **`color c = color(0, 39, 76)`**

Data type

Function

Many Ways to Represent a Color

- `fill(grayscale)`
- `fill(R, G, B)`
- `fill(R, G, B, A)`

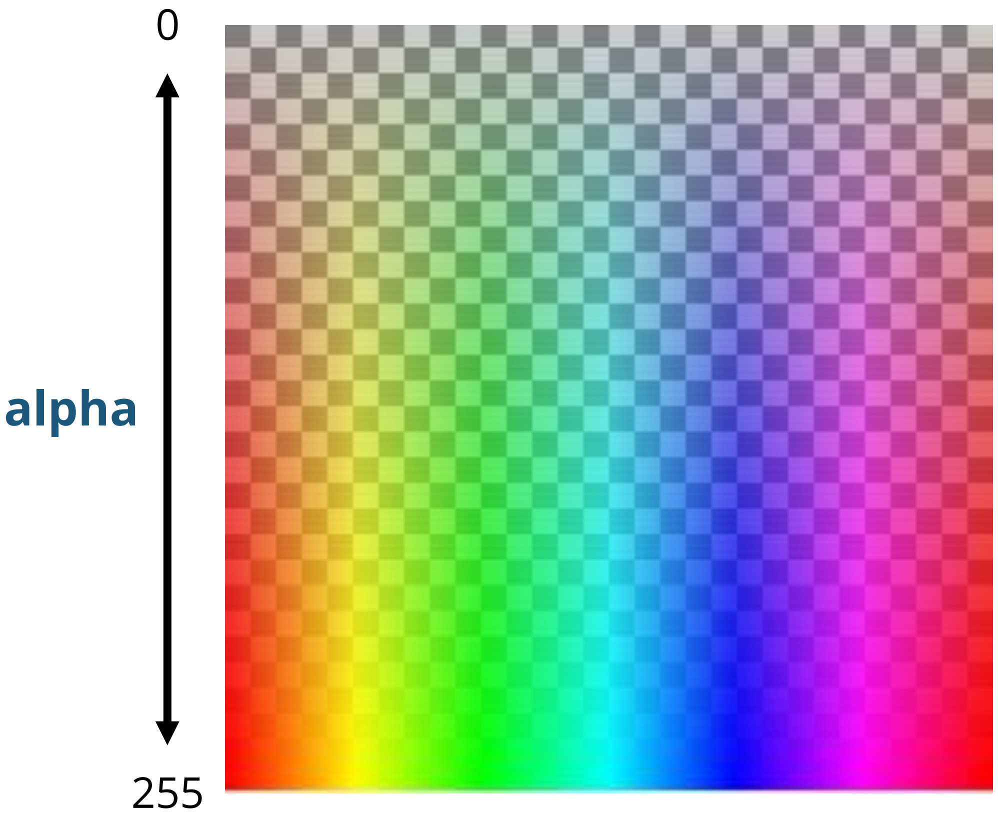
-
  ```
  colorMode(HSB)
  fill(H, S, B)
  ```

-
  ```
  colorMode(HSB)
  fill(H, S, B, A)
  ```

-
  ```
  color c = (0, 39, 76)
  fill(c)
  ```

Transparency – Alpha value

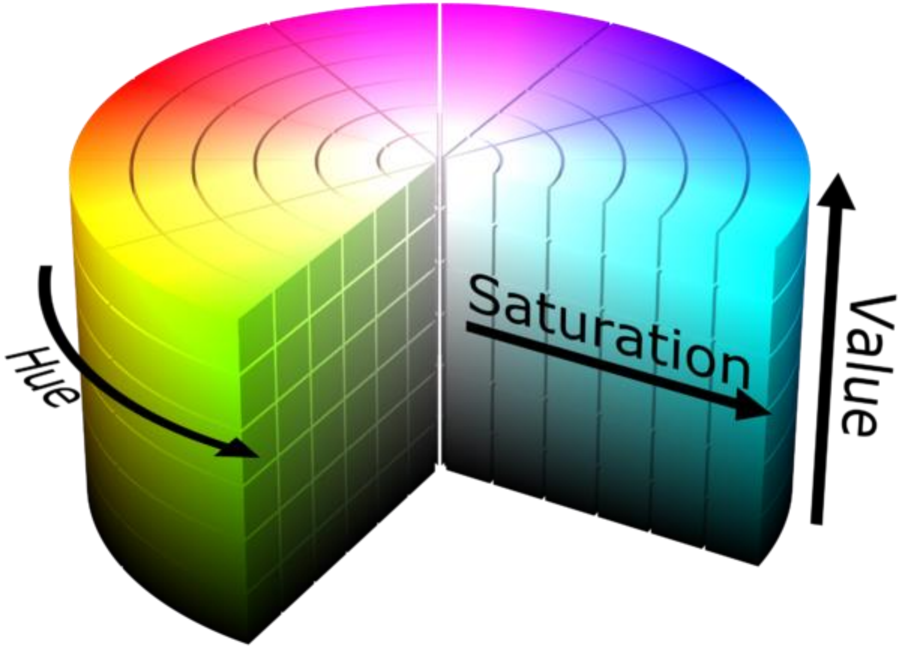
- The alpha value sets the transparency
- Represented as the 4th channel after RGB
  - For example, `fill(0, 39, 76, 50)`
- Useful when you want to **blend** colors



Spitzak, CC BY-SA 4.0, via Wikimedia Commons

HSB Color Mode

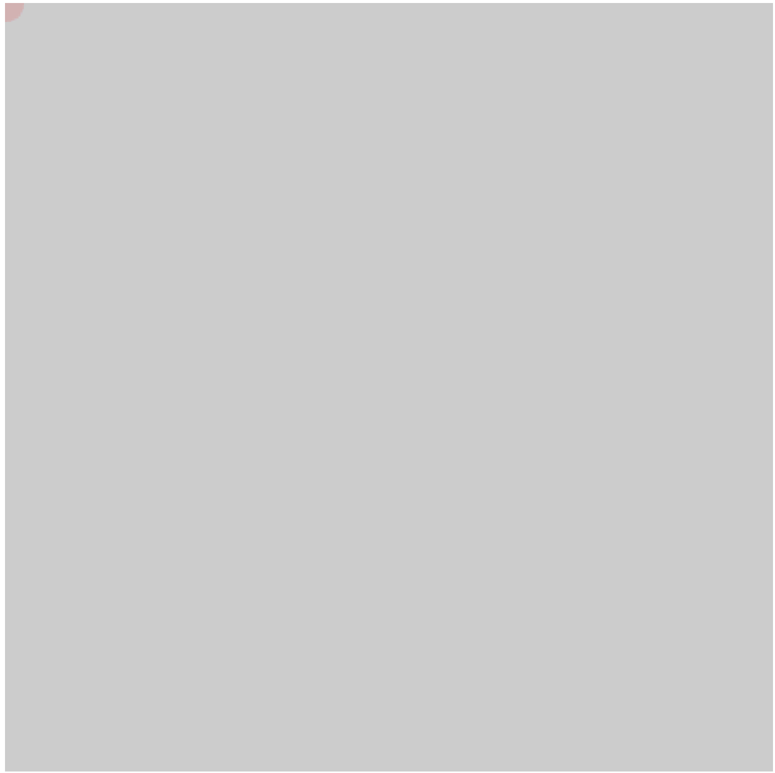
- `colorMode(HSB)`
  - **H** Hue
  - **S** Saturation
  - **B** Brightness
- Useful for creating rainbow effects



SharkD, CC BY-SA 3.0, via Wikimedia Commons

Exercise: Rainbow Paint

- What you'll need:
  - **`mouseX`** & **`mouseY`**
  - **`colorMode(HSB)`** for HSB color mode
  - alpha value for transparency



Fills & Strokes

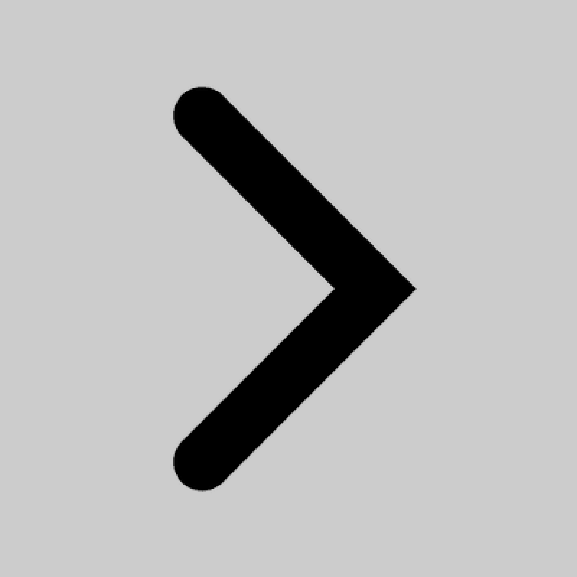
fill() & stroke()

- fill(color)
- noFill()

- stroke(color)
- noStroke()

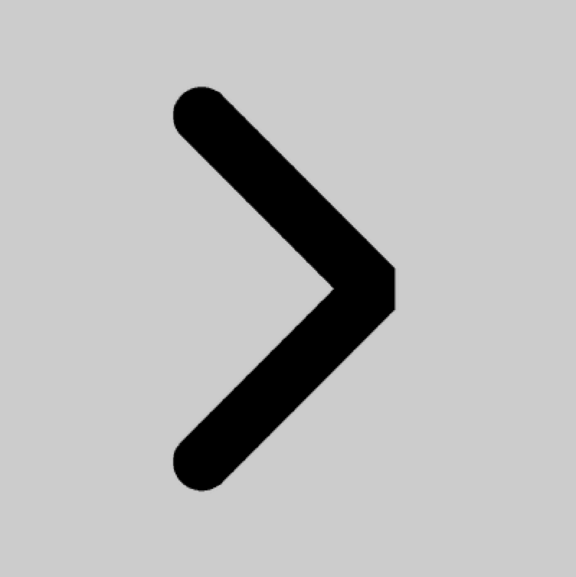
strokeJoin()

strokeJoin(**MITER**)

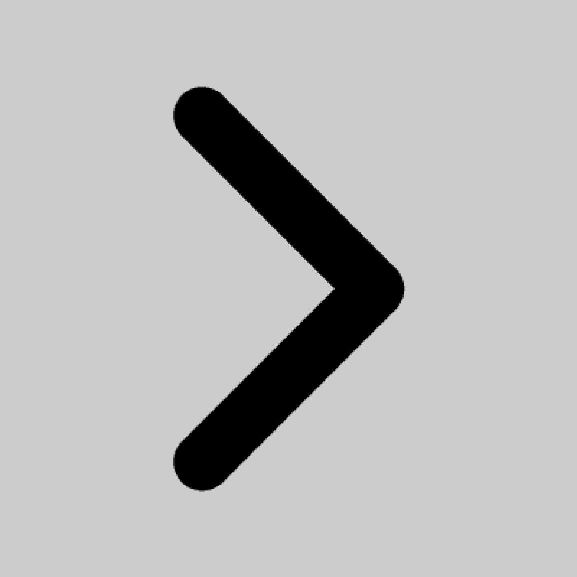


(default)

strokeJoin(**BEVEL**)



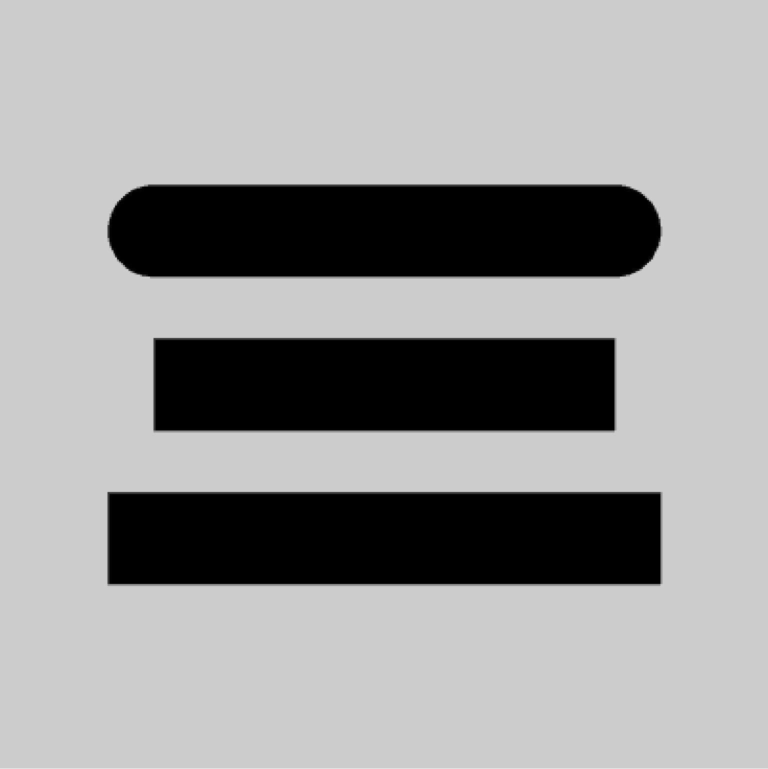
strokeJoin(**ROUND**)



https://processing.org/reference/strokeJoin_.html

strokeCap()

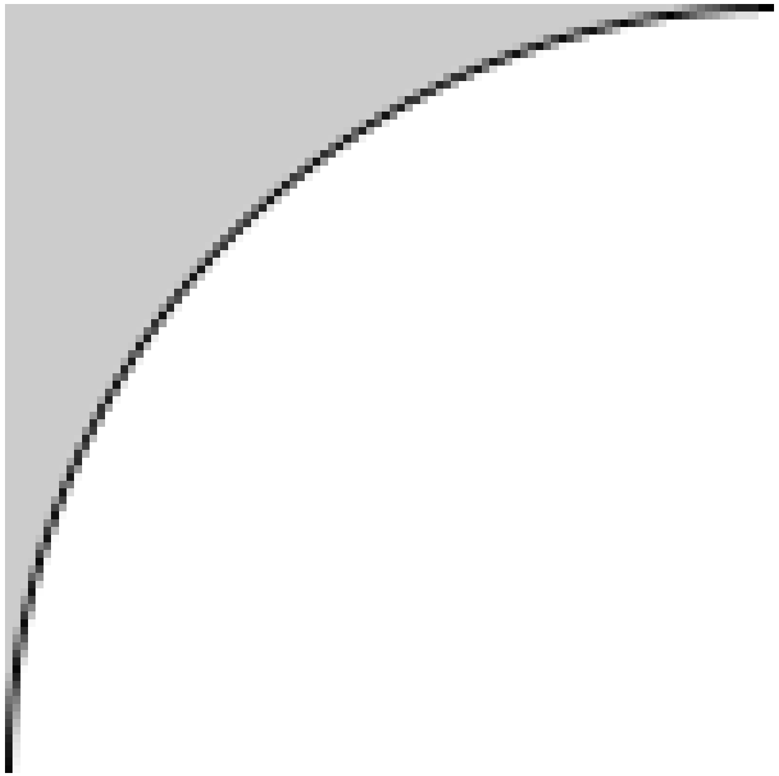
```
strokeCap(ROUND)

strokeCap(SQUARE)

strokeCap(PROJECT)
```



https://processing.org/reference/strokeCap_.html

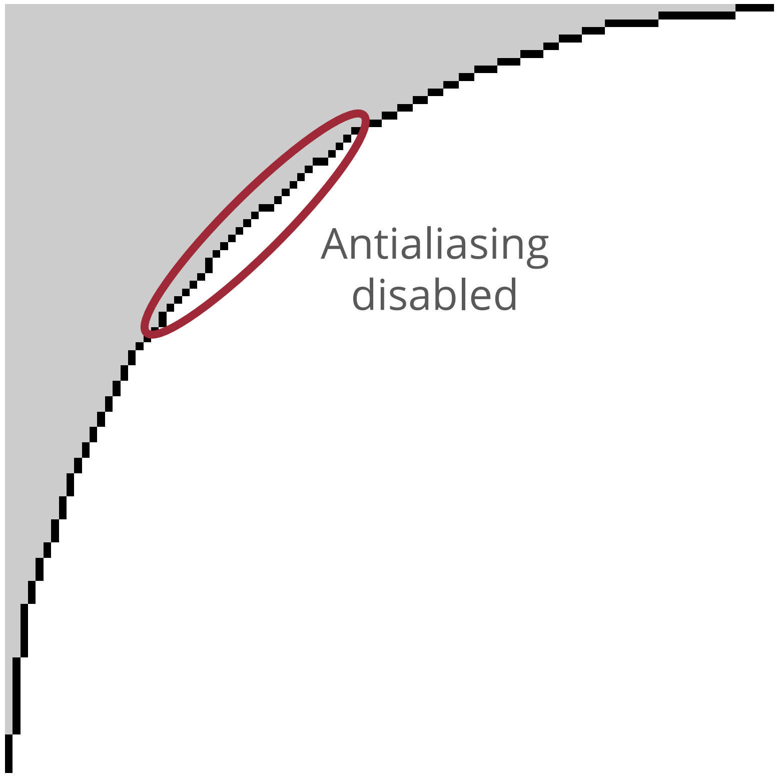
smooth() vs. noSmooth()

`smooth()`



(default)

`noSmooth()`



Antialiasing
disabled