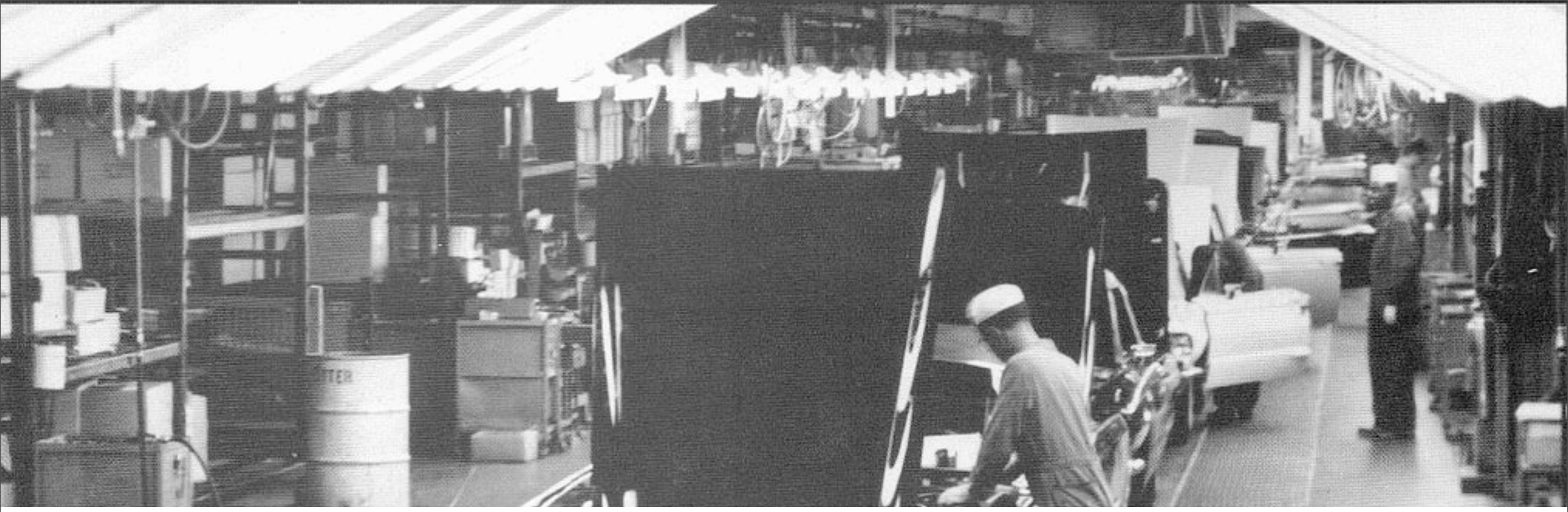
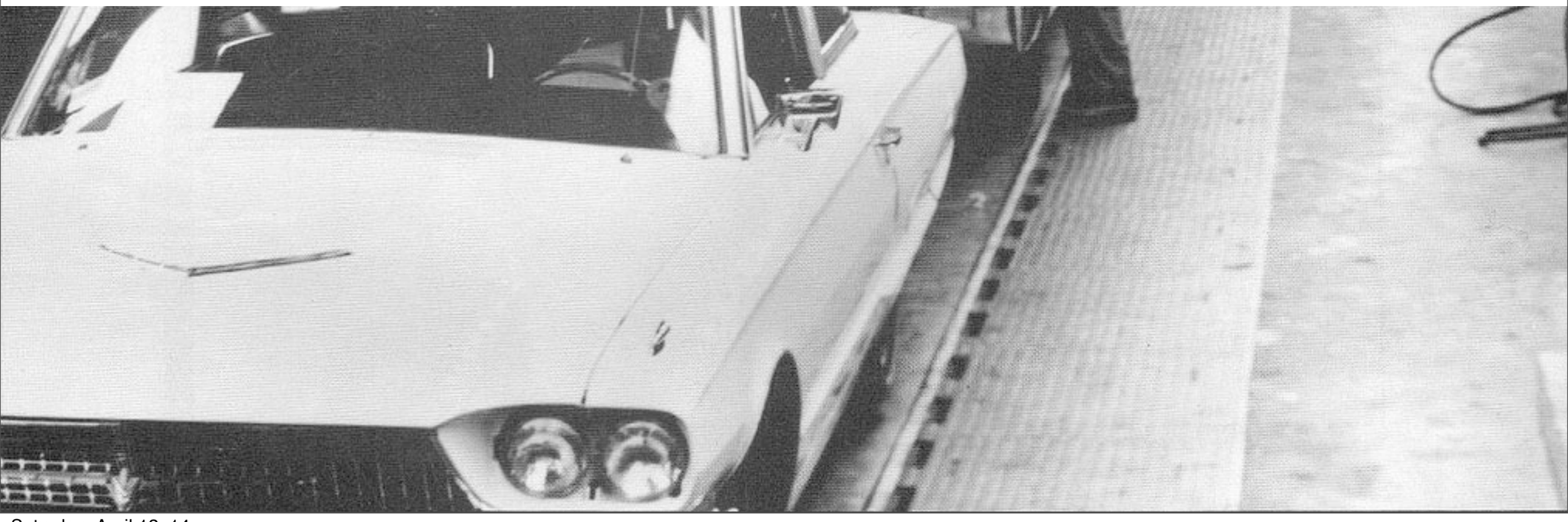
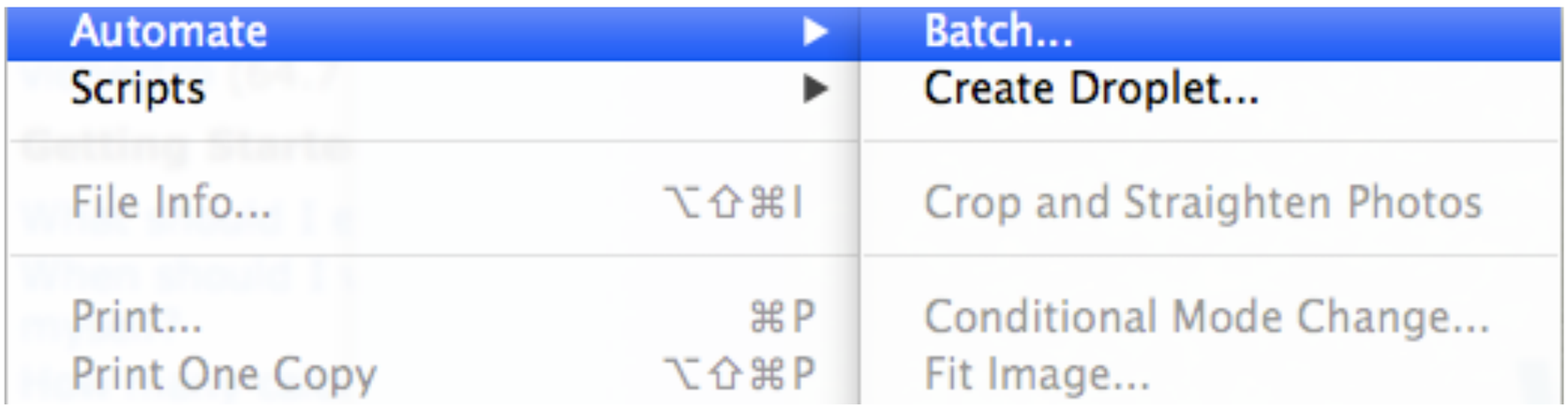


For Loops



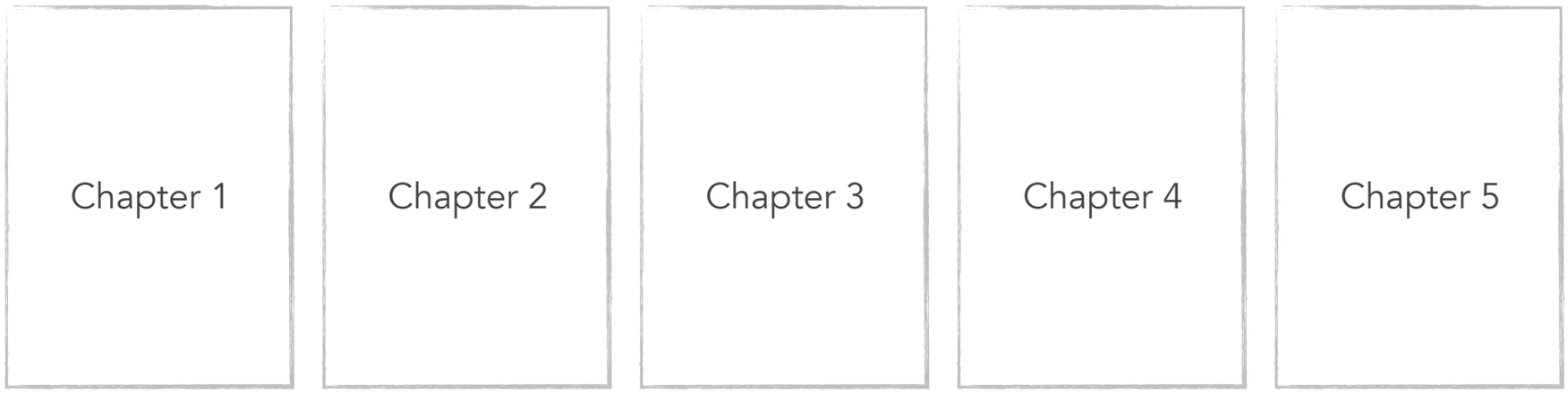
For Loops

- For loops are used to run a chunk of code for a specific number of times
- Compare to the Photoshop “batch” operation--lets you automatically apply the same settings to a certain number of images



For Loops

Let's say we have a book that is **5 chapters** long. Each of those chapters needs to be **proofread**.



How can we break this down into something the computer can understand?

For Loops

Option 1

Proofread chapter 1.
Proofread chapter 2.
Proofread chapter 3.
Proofread chapter 4.
Proofread chapter 5.

Option 2

For each chapter that is in this book, **proofread it**.

→

Start at the **very first chapter**, **proofread it**, then move on to the **next chapter**, **proofread it**, and so on, **until there are no more chapters** to proofread.

For Loops

Start at the **very first chapter**, **proofread it**, then move on to the **next chapter**, **proofread it**, and so on, **until there are no more chapters** to proofread.

→

Start at chapter **i** (here, chapter 1) and proofread it. Add 1 to to **i** and proofread that chapter. Keep adding 1 to **i** until **i** goes past the maximum number of chapters.

For Loops

- In this example, we need to know:
  - The action we want to do (proofread)
  - The total number of times we want to do it (5)
  - How many times we've done it so far (a variable known as an iterator, typically *i*)

For Loops

```
for (int i = 1; i <= 5; i++) {
 println(i);
}
```

`int i = 1;` This is the number we want to start at at the very beginning.

`i <= 5` This is the number we don't want to go past.

`i++` After we perform that action, add 1 to i.

Try running this code and seeing what it does.

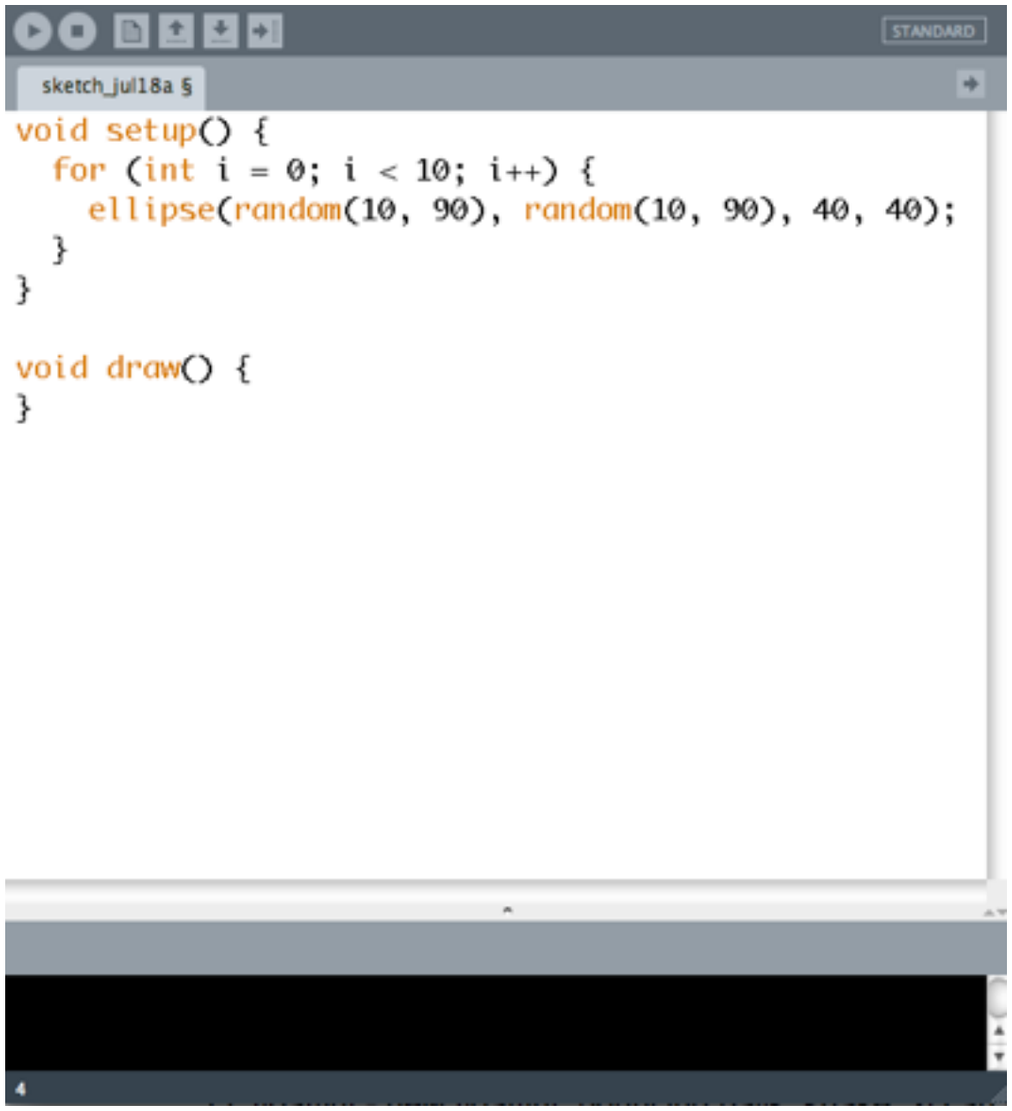
```
for (int i = 1; i <= 5; i++) {
  println(i);
}
```

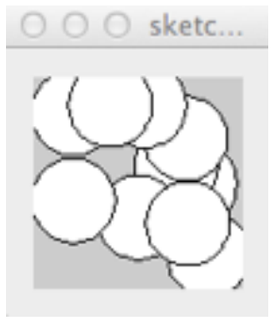
For Loop Written Out

```
for (int i = 1; i <= 5; i++) {
 println(i);
}
```

i	Less than or equal to 5?	Performs action?	Increments?
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	Yes	Yes
4	Yes	Yes	Yes
5	Yes	Yes	Yes
6	No	No	No

Try drawing 10 ellipses at random x- and y-positions using a for loop.





We'll discuss why programmers start counting with 0 when we get to arrays.

Compare.

STANDARD

sketch_jul18a §

```
void setup() {
  for (int i = 0; i < 10; i++) {
    ellipse(random(10, 90), random(10, 90), 40, 40);
  }
}

void draw() {
}
```

4

STANDARD

sketch_jul18a §

```
void setup() {
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
  ellipse(random(10, 90), random(10, 90), 40, 40);
}

void draw() {
}
```

Auto Format finished.

16

Arrays

Arrays

- So far, we've been creating variables individually and separately, giving each a unique name
- Some would call this "super annoying"

```
int circle1_xPos = random(width);
int circle2_xPos = random(width);
int circle3_xPos = random(width);
```

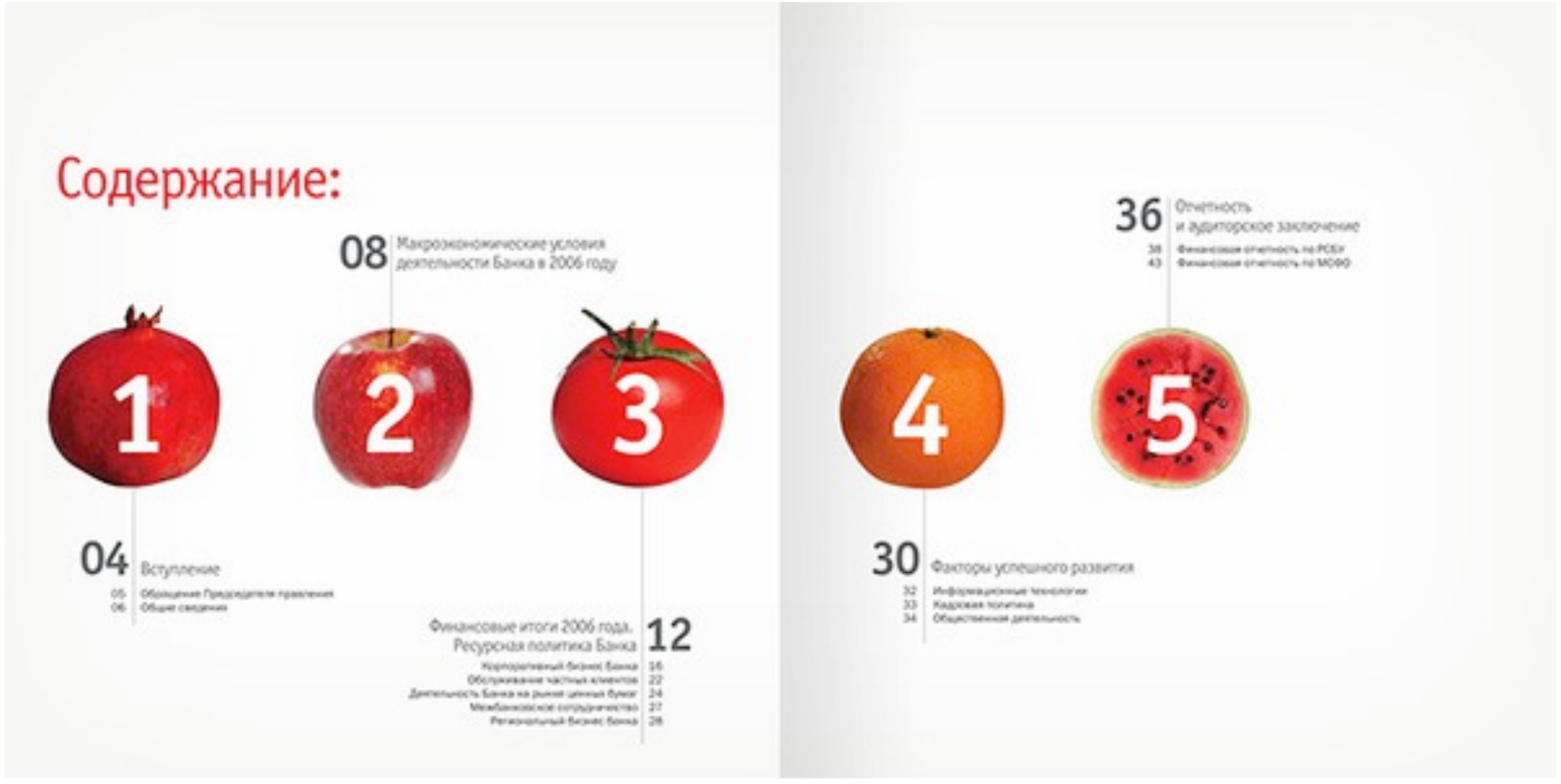
Arrays

- Arrays let us group similar variables together
- Array as a whole has a name; individual variables are referenced by their *index*

```
int[] circle_xPositions = new int[3];
```

Arrays

- It's a bit like how a book holds chapters, and gives each chapter a reference number



Arrays

```
int[] circle_xPositions = new int[3];
```

First, we say what type of data we want our array to hold. Then we put open and close brackets to indicate that it's an **array of ints**, not just one int.

(Think of the brackets like a box encircling multiple pieces of data.)

Arrays

```
int[] circle_xPositions = new int[3];
```


Next, we name our array--just like how we name any other variable.

Going along with the example before, we'll name it *circle_xPositions*, since we want it to hold multiple circle x-positions.

Arrays

```
int[] circle_xPositions = new int[3];
```

Here, all we're really saying is that Processing should go ahead and make us this new array.

Arrays

```
int[] circle_xPositions = new int[3];
```

Processing needs to know how many slots for information this array needs to have, so we indicate that it needs 3 slots, which will be holding integers.

(If you want your array to be able to change its length, you use something called an arraylist, which we won't be covering today.)

Arrays

```
int[] circle_xPositions = new int[3];
```

Summarized: we have this variable called *circle_xPositions*, which is an array of integers. That array should be created with 3 spots to hold integer data.

Setting and Retrieving Data

- Two ways to set initial values: { }, or [] (see below)
- You can retrieve or change information in a slot by putting its reference number in those square brackets

```
float[] circle_xPositions = {30, 60, 90};


circle_xPositions[0] = random(width);
circle_xPositions[1] = random(width);
circle_xPositions[2] = random(width);
```

Zero Indexing

```
circle_xPositions[0] = random(width);
circle_xPositions[1] = random(width);
circle_xPositions[2] = random(width);
```

Zero Indexing - Short Answer

That's just the way earlier programmers developed it. Take it up with them.

Zero Indexing - Actual Answer

- When you create a variable, Processing sets aside space in memory to hold it.
- Each spot in memory has an address, like a house on a street.

Houses

206	**207**	**208**	**209**
Lannisters	Starks	Targaryens	Tyrells

Memory Addresses

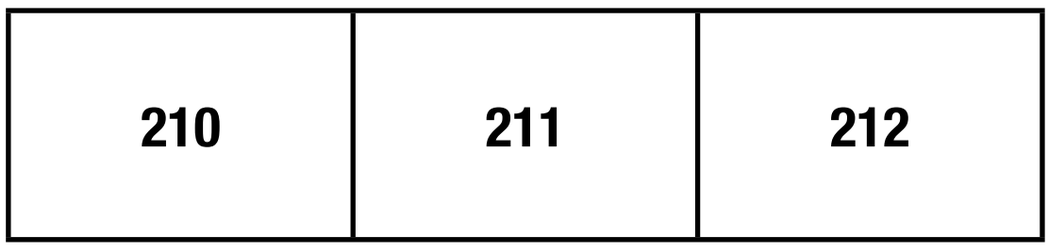
206	**207**	**208**	**209**
“Hello”	42	18.5	2PI

actual memory addresses are way longer--e.g. 0x8130--but you get the idea

Zero Indexing - Actual Answer

When you create an array, Processing sets aside **consecutive slots of memory**--as many slots as you asked for when you created the array.

```
int[] circle_xPositions = new int[3];
```

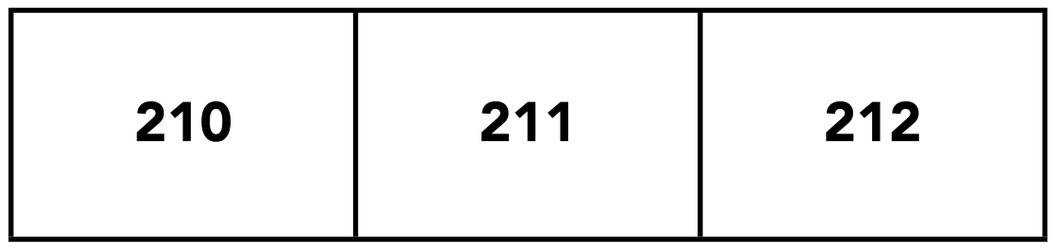


Zero Indexing - Actual Answer

The way Processing thinks about arrays is:

"Okay, this array **begins** at 210. Its **length** is 3. So any slot in this array can be thought about as **some distance away from 210**."

```
int[] circle_xPositions = new int[3];
```



Zero Indexing - Actual Answer

How far away is slot 210 from slot 210?

210	211	212

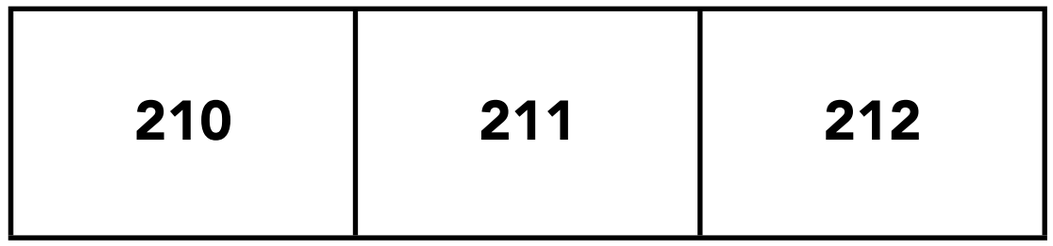
(Not a trick question.)

Zero Indexing - Actual Answer

210 **is** the first slot, so it is **zero** distance away from the beginning.

211 is the second slot, and it's **one** slot away from the beginning.

212 is the third slot, and it's **two** slots away from the beginning.

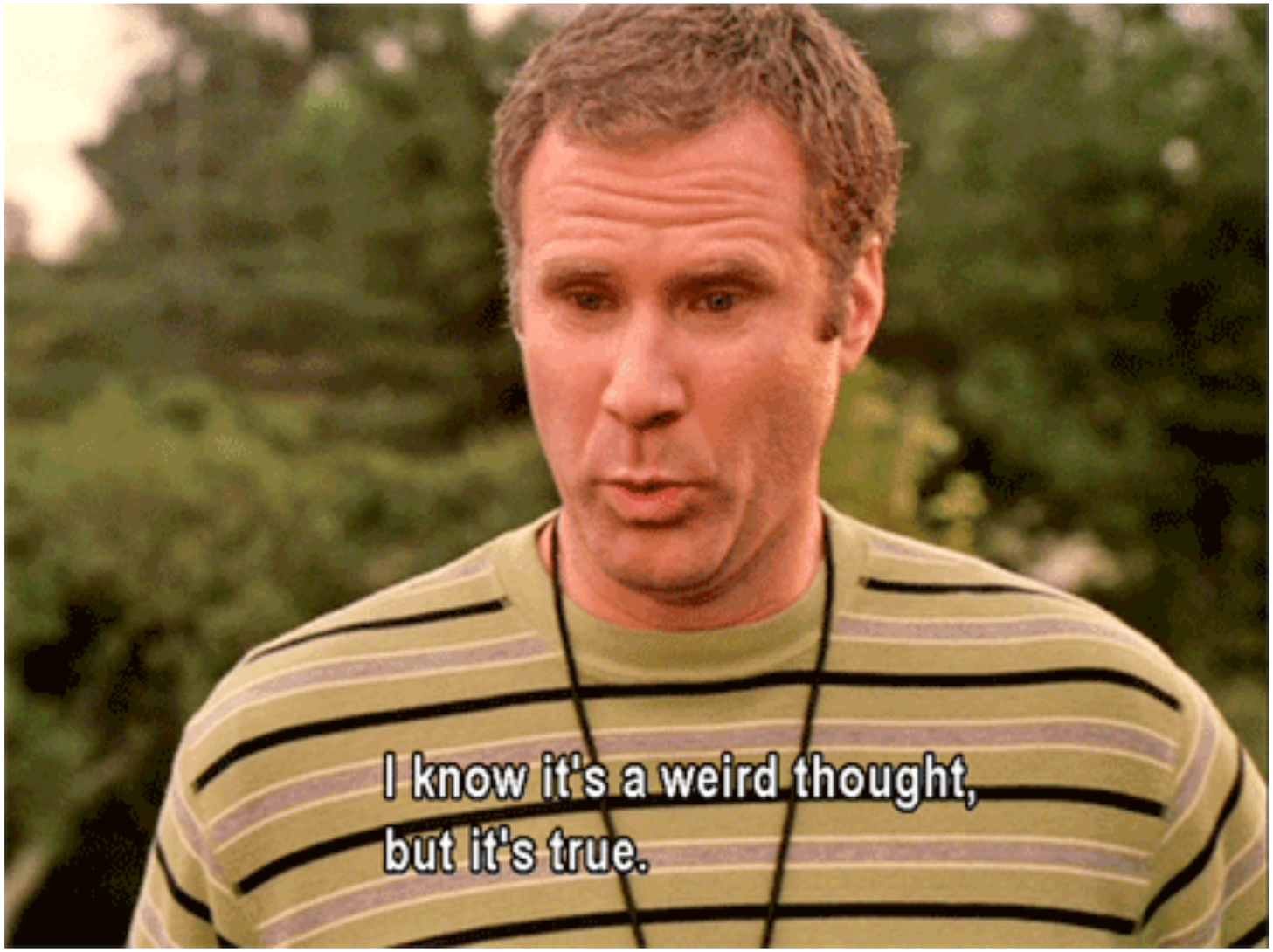


Similarly, the first slot in the array of *circle_xPositions* is **zero** distance away from the beginning. It **is** the beginning!

The second slot is **one** slot away from the beginning.

The third slot is **two** slots away from the beginning.

```
circle_xPositions[0] = random(width);
circle_xPositions[1] = random(width);
circle_xPositions[2] = random(width);
```


I know it's a weird thought,
but it's true.

Zero Indexing - Calm Down

Don't worry about internalizing all that right now. As long as you remember to start your arrays from 0, you're fine.

Let's say we want to have 5 balls on screen that are located at random y-locations.

Make an array of floats to hold 5 circle y-locations.

Don't worry about filling it yet. We'll do that next.

```
float[] circle_yPositions = new float[5];
```

Now go through each slot in the array, and set it to a random value between 0 and the height of the sketch.

STANDARD

sketch_jul25a §

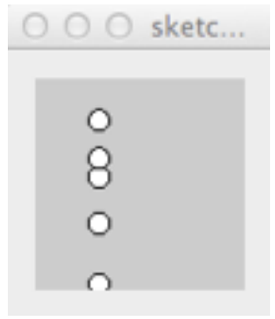
```
float[] circle_yPositions = new float[5];

void setup() {
  circle_yPositions[0] = random(height);
  circle_yPositions[1] = random(height);
  circle_yPositions[2] = random(height);
  circle_yPositions[3] = random(height);
  circle_yPositions[4] = random(height);
}

void draw() {
  ellipse(30, circle_yPositions[0], 10, 10);
  ellipse(30, circle_yPositions[1], 10, 10);
  ellipse(30, circle_yPositions[2], 10, 10);
  ellipse(30, circle_yPositions[3], 10, 10);
  ellipse(30, circle_yPositions[4], 10, 10);
}
```

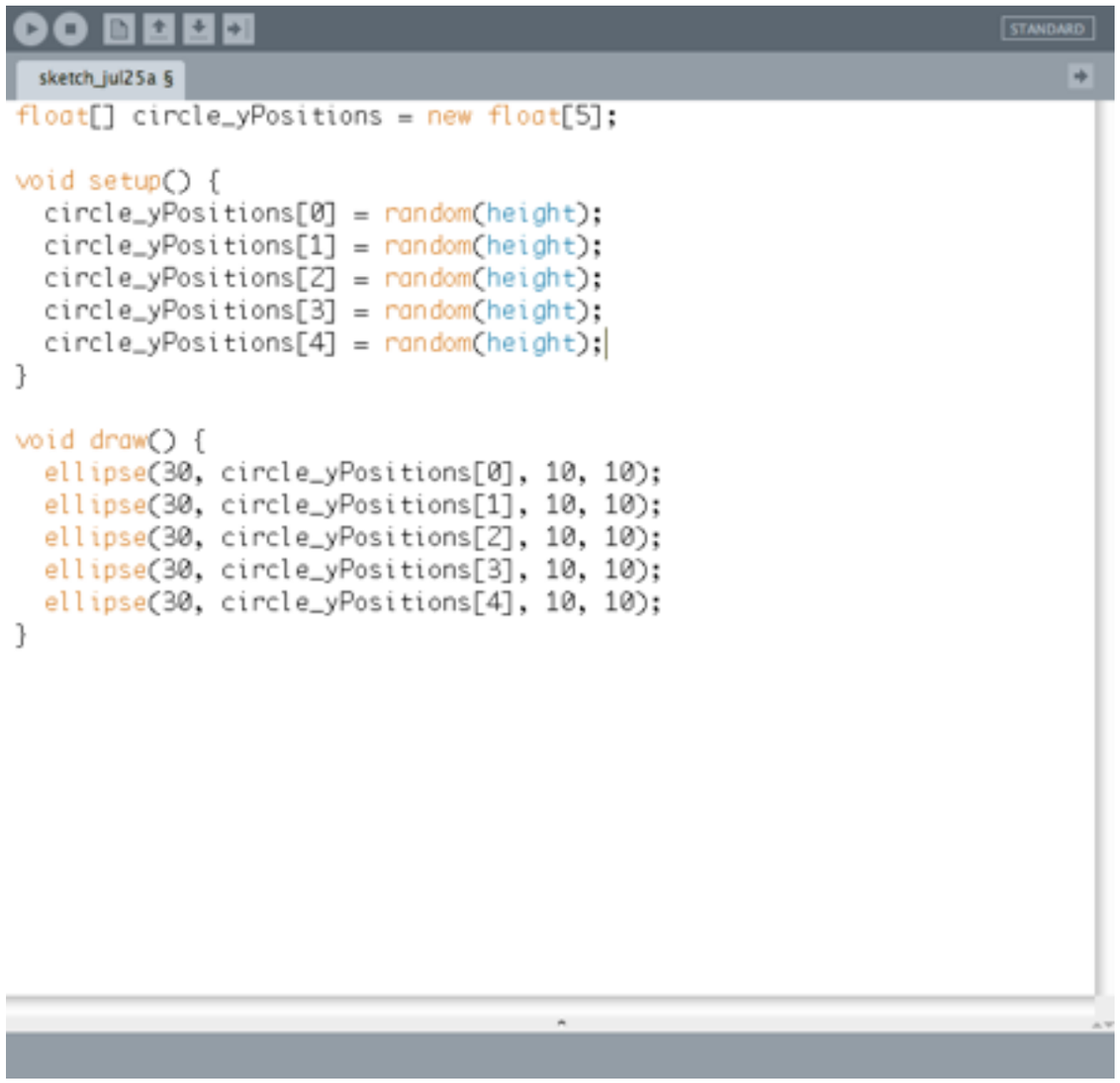
Now try drawing 5 circles, with each one using the y-position in one of the slots.

STANDARD

sketch_jul25a §

```
float[] circle_yPositions = new float[5];

void setup() {
  circle_yPositions[0] = random(height);
  circle_yPositions[1] = random(height);
  circle_yPositions[2] = random(height);
  circle_yPositions[3] = random(height);
  circle_yPositions[4] = random(height);
}

void draw() {
  ellipse(30, circle_yPositions[0], 10, 10);
  ellipse(30, circle_yPositions[1], 10, 10);
  ellipse(30, circle_yPositions[2], 10, 10);
  ellipse(30, circle_yPositions[3], 10, 10);
  ellipse(30, circle_yPositions[4], 10, 10);
}
```





Can you remember a concept we learned that allowed us to execute an action over and over again?

```
for (int i = 1; i <= 5; i++) {
 println(i);
}
```

For-loop

```
for (int i = 1; i <= 5; i++) {
 println(i);
}
```

Array

```
circle_yPositions[0] = random(height);
circle_yPositions[1] = random(height);
circle_yPositions[2] = random(height);
circle_yPositions[3] = random(height);
circle_yPositions[4] = random(height);
```

Do you remember what **i** does?

Can you see how it might relate to what's happening in our *circle_yPositions* brackets?

Inefficient

```
circle_yPositions[0] = random(height);
circle_yPositions[1] = random(height);
circle_yPositions[2] = random(height);
circle_yPositions[3] = random(height);
circle_yPositions[4] = random(height);
```

Efficient

```
for (int i = 0; i < 5; i++) {
 circle_yPositions[i] = random(height);
}
```

When i=0, we're dealing with circle_yPositions[0].
When i increments (i=1), we're dealing with circle_yPositions[1].
And so on.
Note that **i** should be equal to the length of your array. Otherwise, you'll either go outside your array, or leave elements out.

Can you write a for-loop that will go through each element in *circle_yPositions* and draw a circle at each of those y-positions?

```
for (int i = 0; i < 5; i++) {
  ellipse(30, circle_yPositions[i], 10, 10);
}
```

Compare.

STANDARD

sketch_jul25a §

```
float[] circle_yPositions = new float[5];

void setup() {
  circle_yPositions[0] = random(height);
  circle_yPositions[1] = random(height);
  circle_yPositions[2] = random(height);
  circle_yPositions[3] = random(height);
  circle_yPositions[4] = random(height);
}

void draw() {
  ellipse(30, circle_yPositions[0], 10, 10);
  ellipse(30, circle_yPositions[1], 10, 10);
  ellipse(30, circle_yPositions[2], 10, 10);
  ellipse(30, circle_yPositions[3], 10, 10);
  ellipse(30, circle_yPositions[4], 10, 10);
}
```

STANDARD

sketch_jul25a §

```
float[] circle_yPositions = new float[5];

void setup() {
  for (int i = 0; i < 5; i++) {
    circle_yPositions[i] = random(height);
  }
}

void draw() {
  for (int i = 0; i < 5; i++) {
    ellipse(30, circle_yPositions[i], 10, 10);
  }
}
```

Auto Format finished.

Can you alter that loop so that each frame, we subtract .25 from each float held by *circle_yPositions?*

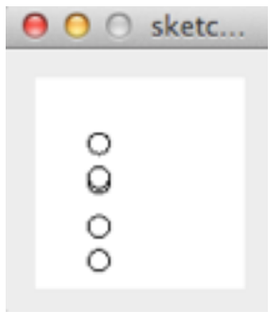
It should only be 1 additional line of code.

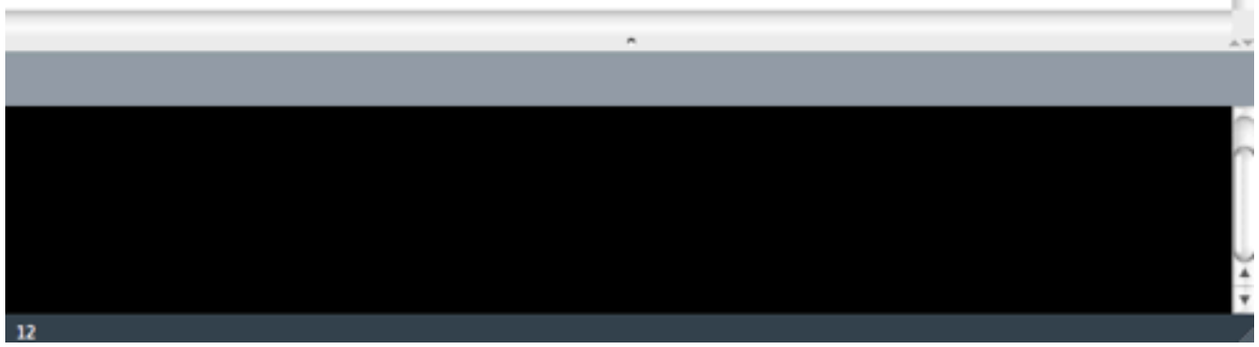
But you may want to add background(255); to the top of your draw loop.

STANDARD

sketch_jul25a §

```
float[] circle_yPositions = new float[5];

void setup() {
  for (int i = 0; i < 5; i++) {
    circle_yPositions[i] = random(height);
  }
}

void draw() {
  background(255);
  for (int i = 0; i < 5; i++) {
    circle_yPositions[i] = circle_yPositions[i] - .25;
    ellipse(30, circle_yPositions[i], 10, 10);
  }
}
```





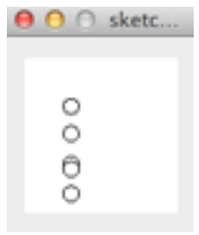
Let's say you don't want your circles to disappear forever. Say we want them to reverse direction once they hit the top or bottom of the screen. And let's say we want them to have different y velocities (from 1 to 3).

Can you combine arrays, if-statements, and for-loops to make a sketch that has the balls bounce up and down?

sketch_aug06g §

```
float[] circle_yPositions = new float[5];
float[] circle_yVelocities = new float[5];

void setup() {
  for (int i = 0; i < 5; i++) {
    circle_yPositions[i] = random(height);
  }

  for (int i = 0; i < 5; i++) {
    circle_yVelocities[i] = random(1, 3);
  }
}

void draw() {
  background(255);
  for (int i = 0; i < 5; i++) {
    if (circle_yPositions[i] < 0 || circle_yPositions[i] > height) {
      circle_yVelocities[i]*=-1;
    }

    circle_yPositions[i]+=circle_yVelocities[i];
    ellipse(30, circle_yPositions[i], 10, 10);
  }
}
```

22



Mouse and Keyboard Input

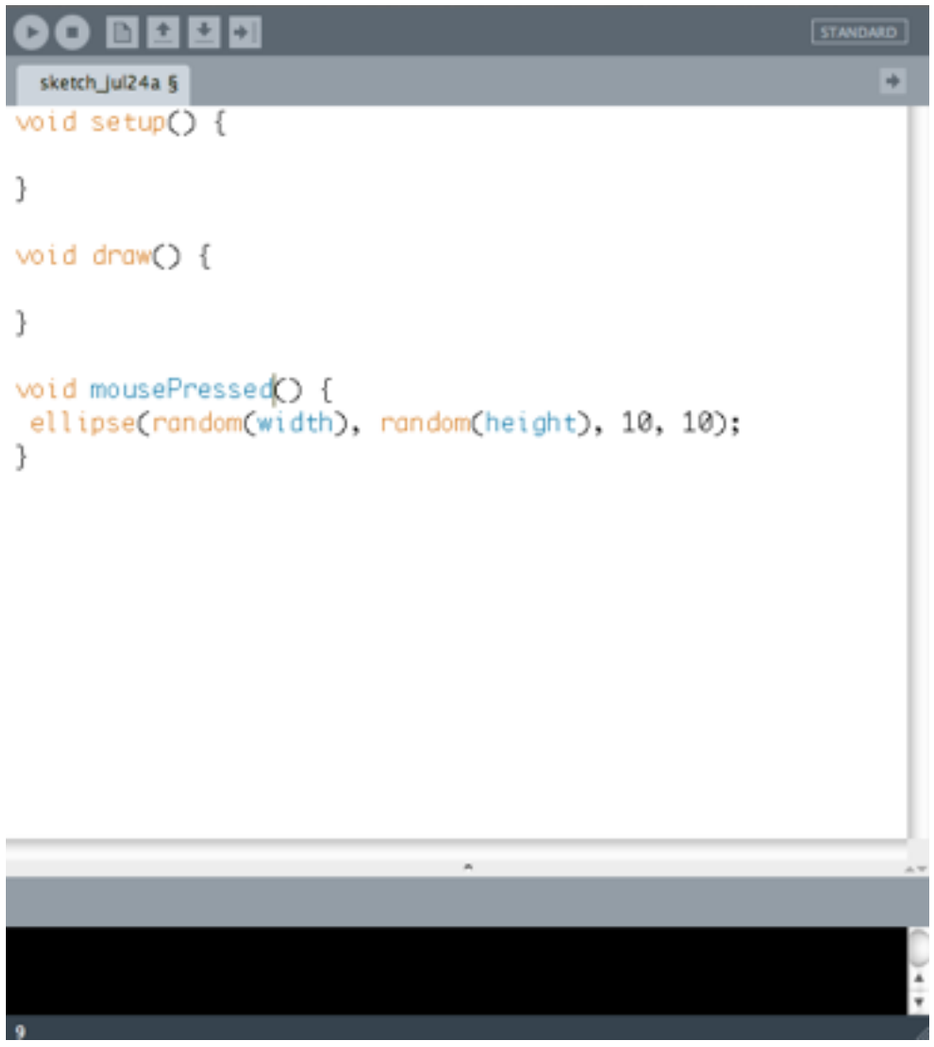
Built-In Functions - Mouse

- Functions: modular pieces of code that can be referenced by name
  - ellipse()
  - draw()
- Processing runs some functions automatically whenever a given event happens
- You can invoke those functions and put your own code inside

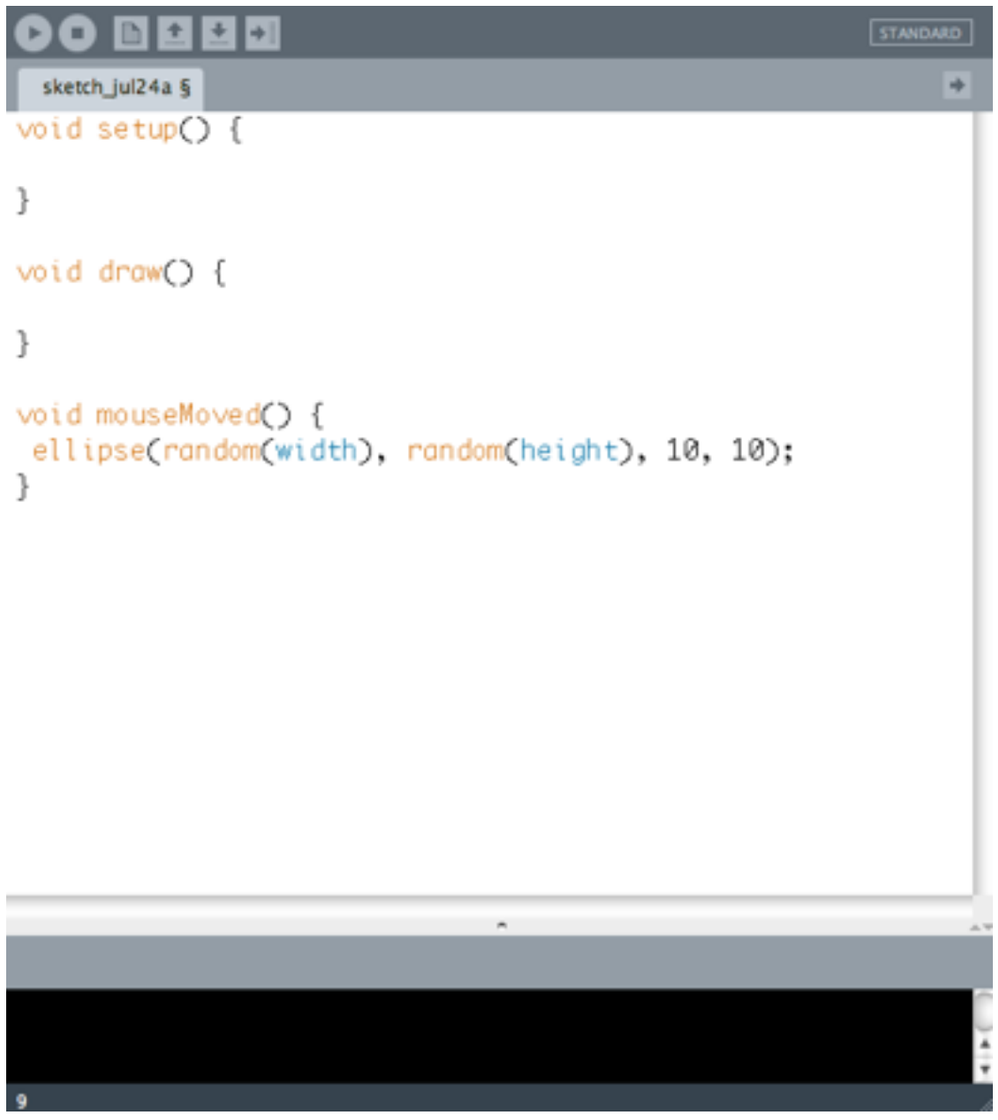
Built-In Functions - Mouse

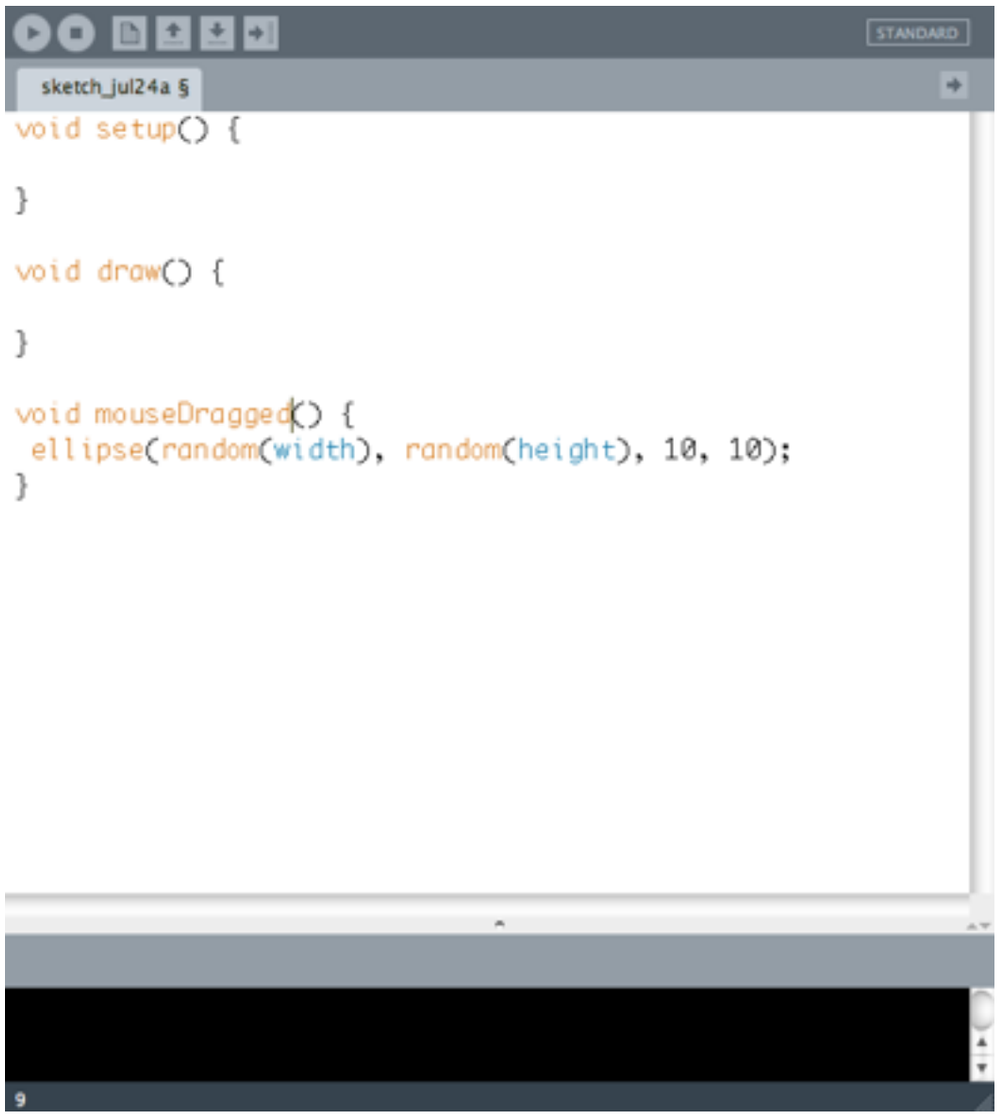
Function	When is it run?
void mousePressed() { }	After the mouse button is pressed down (but just once for each press).
void mouseReleased() { }	After the mouse is released (but just once after each release).
void mouseMoved() { }	Every frame that the mouse is moved (and a button is NOT pressed).
void mouseDragged() { }	Every frame that the mouse is moved (and a button IS pressed).
void mouseClicked() { }	After the mouse button is pressed and released (but just once for each click).

Try running these pieces of code. How do they operate differently?



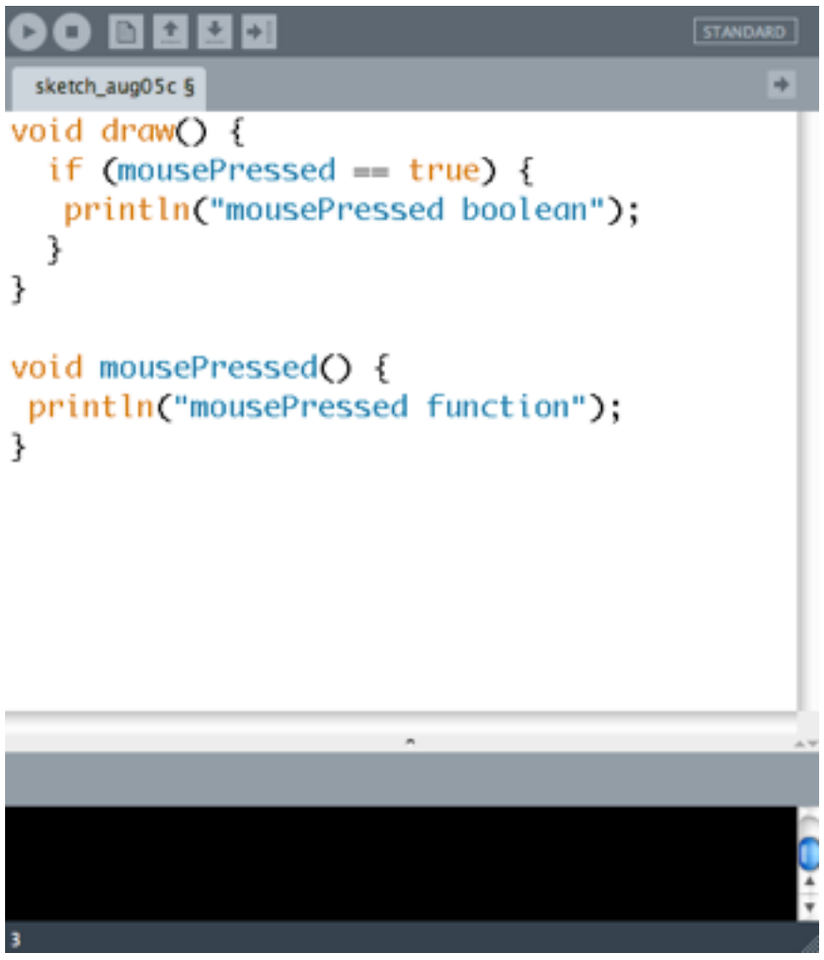
How about these?





Built-In Functions - Mouse

- Mouse pressing comes in two flavors:
  - Boolean (true for the whole time the mouse is pressed)
  - Function (true for the first frame that the mouse is pressed)



Try using mouseDragged() to draw a line between the current position of the mouse, and the position of the mouse last frame.

STANDARD

sketch_jul24a §

```
void setup() {

}

void draw() {

}

void mouseDragged() {
 line(mouseX, mouseY, pmouseX, pmouseY);
}
```

10

Try jazzing up that last sketch by using different colors, different backgrounds, etc.

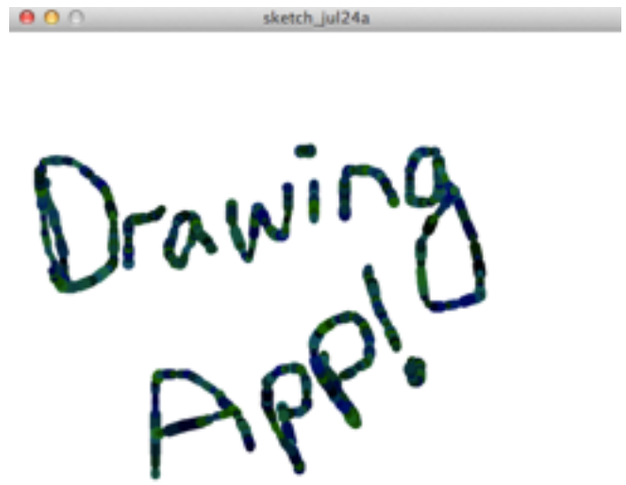
Some suggestions: try using fill(), strokeWeight(), and smooth().

STANDARD

sketch_jul24a §

```
void setup() {
  size(500, 500);
  smooth();
  background(255);
}

void draw() {
  strokeWeight(random(5, 10));
}

void mouseDragged() {
  stroke(0, random(100), random(100));
  line(mouseX, mouseY, pmouseX, pmouseY);
}
```

1



What if you wanted the color to change each time the user started drawing a new line?

Hint: the user starts drawing a line when the mouse is pressed down.

STANDARD

sketch_jul24a §

```
void setup() {
  size(500, 500);
  smooth();
  background(255);
}

void draw() {
  strokeWeight(random(5, 10));
}

void mouseDragged() {
  line(mouseX, mouseY, pmouseX, pmouseY);
}

void mousePressed() {
  stroke(0, random(100), random(100));
}
```

18

Built-In Functions - Keys

Called and run automatically, given a certain condition.

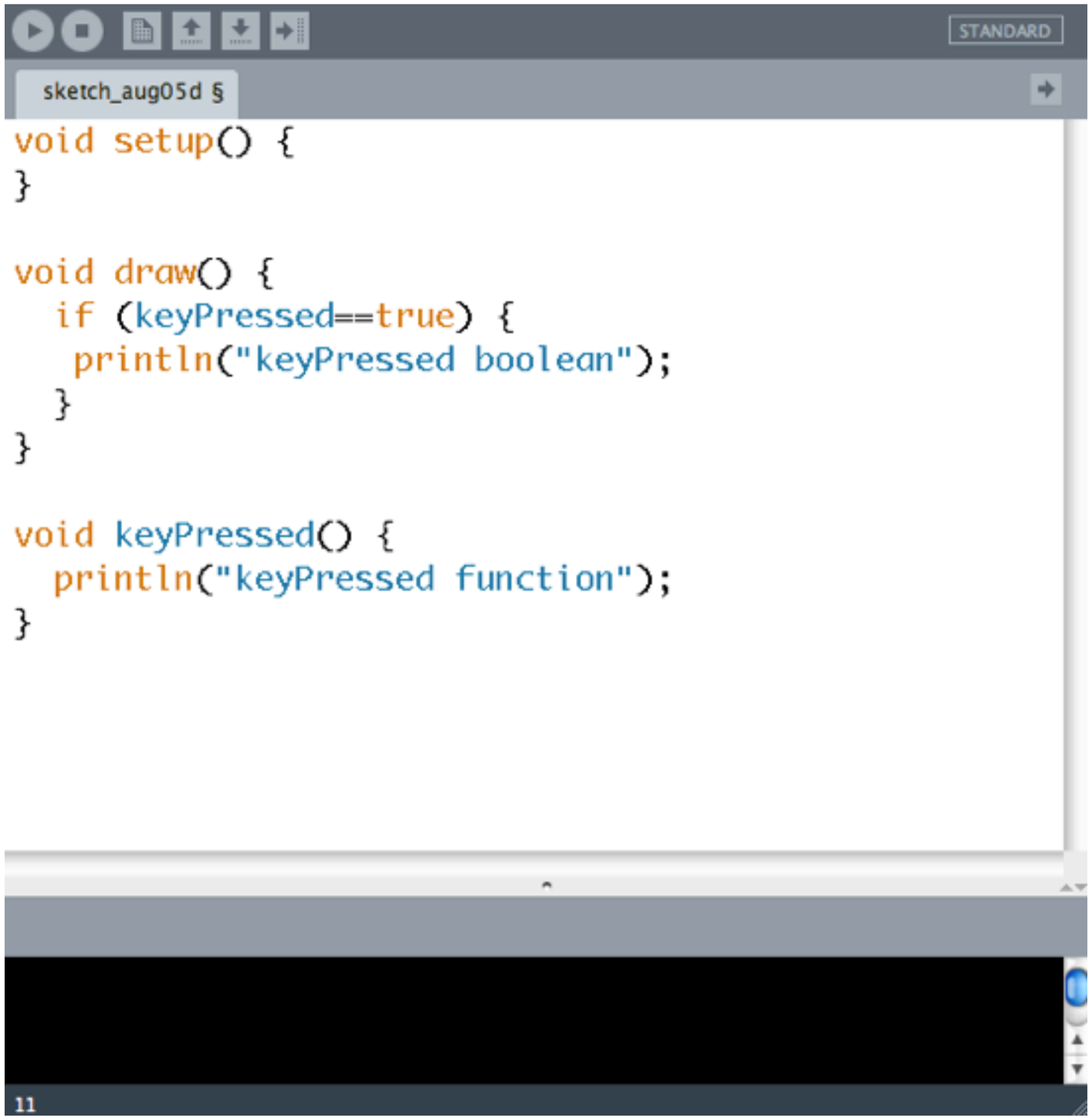
Function	When is it run?
void keyPressed() { }	After a key is pressed.
void keyReleased() { }	After a key is released.

Specifying Letter Keys

- To specify letters, you can use:
  - The number associated with each key (ASCII)
  - The letter itself (e.g. ‘a’)

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
32	040	20	00100000		&#32;		Space
33	041	21	00100001	!	&#33;		Exclamation mark
34	042	22	00100010	"	&#34;	&quot;	Double quotes (or speech marks)
35	043	23	00100011	#	&#35;		Number
36	044	24	00100100	$	&#36;		Dollar
37	045	25	00100101	%	&#37;		Procenttecken
38	046	26	00100110	&	&#38;	&amp;	Ampersand
39	047	27	00100111	'	&#39;		Single quote
40	050	28	00101000	(	&#40;		Open parenthesis (or open bracket)
41	051	29	00101001	)	&#41;		Close parenthesis (or close bracket)
42	052	2A	00101010	*	&#42;		Asterisk
43	053	2B	00101011	+	&#43;		Plus
44	054	2C	00101100	,	&#44;		Comma
45	055	2D	00101101	-	&#45;		Hyphen
46	056	2E	00101110	.	&#46;		Period, dot or full stop
47	057	2F	00101111	/	&#47;		Slash or divide
48	060	30	00110000	0	&#48;		Zero
49	061	31	00110001	1	&#49;		One
50	062	32	00110010	2	&#50;		Two
51	063	33	00110011	3	&#51;		Three
52	064	34	00110100	4	&#52;		Four
53	065	35	00110101	5	&#53;		Five
54	066	36	00110110	6	&#54;		Six
55	067	37	00110111	7	&#55;		Seven
56	070	38	00111000	8	&#56;		Eight
57	071	39	00111001	9	&#57;		Nine
58	072	3A	00111010	:	&#58;		Colon
59	073	3B	00111011	;	&#59;		Semicolon
60	074	3C	00111100	<	&#60;	&lt;	Less than (or open angled bracket)
61	075	3D	00111101	=	&#61;		Equals
62	076	3E	00111110	>	&#62;	&gt;	Greater than (or close angled bracket)
63	077	3F	00111111	?	&#63;		Question mark
64	100	40	01000000	@	&#64;		At symbol
65	101	41	01000001	A	&#65;		Uppercase A
66	102	42	01000010	B	&#66;		Uppercase B
67	103	43	01000011	C	&#67;		Uppercase C
68	104	44	01000100	D	&#68;		Uppercase D
69	105	45	01000101	E	&#69;		Uppercase E

```
void keyPressed() {
 if (key == 99) {
  ellipse(random(width), random(height), 10, 10);
 }
}
```


STANDARD
sketch_aug05d §
void setup() {
}
void draw() {
if (keyPressed==true) {
println("keyPressed boolean");
}
}
void keyPressed() {
println("keyPressed function");
}
11

Try writing a sketch where pressing the letter ‘c’ will draw a circle at a random point, and pressing the letter ‘r’ will draw a rectangle at a random point.

Hint: the key values for uppercase and lowercase values are different. Make sure you’re using the right one.

Save STANDARD

sketch_jul24a §

```
void setup() {
  size(500, 500);
  smooth();
  background(255);
}

void draw() {

}

void keyPressed() {
 if (key == 99) {
  ellipse(random(width), random(height), 10, 10);
 } else if (key == 114) {
  rect(random(width), random(height), 10, 10);
 }
}
```

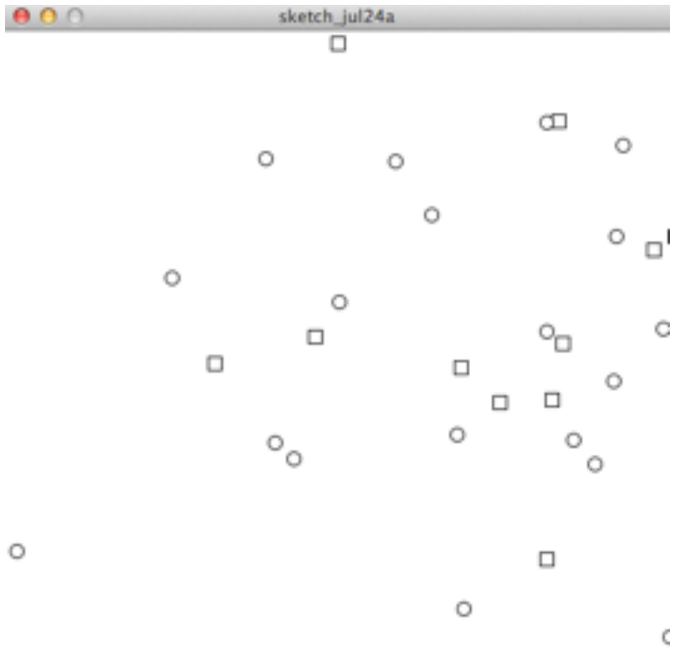
15

STANDARD

sketch_jul24a §

```
void setup() {
  size(500, 500);
  smooth();
  background(255);
}

void draw() {

}

void keyPressed() {
 if (key == 99) {
  ellipse(random(width), random(height), 10, 10);
 }

 if (key == 114) {
  rect(random(width), random(height), 10, 10);
 }
}
```

16



Specifying Non-Letter Keys

```
void keyPressed() {
  if (keyCode == UP) {
    yPos-=1;
  }

  if (keyCode == DOWN) {
    yPos+=1;
  }
}
```

STANDARD

sketch_jul24a §

```
float yPos;

void setup() {
  size(500, 500);
  smooth();

  yPos = height/2;
}

void draw() {
  background(255);
  ellipse(width/2, yPos, 10, 10);
}

void keyPressed() {
  if (keyCode == UP) {
    yPos-=1;
  }

  if (keyCode == DOWN) {
    yPos+=1;
  }
}
```

17

Time

Millis

- Millis(): returns the amount of time that has passed since the app started in milliseconds
- Useful for creating countdown timers, setting things to happen on intervals, etc.