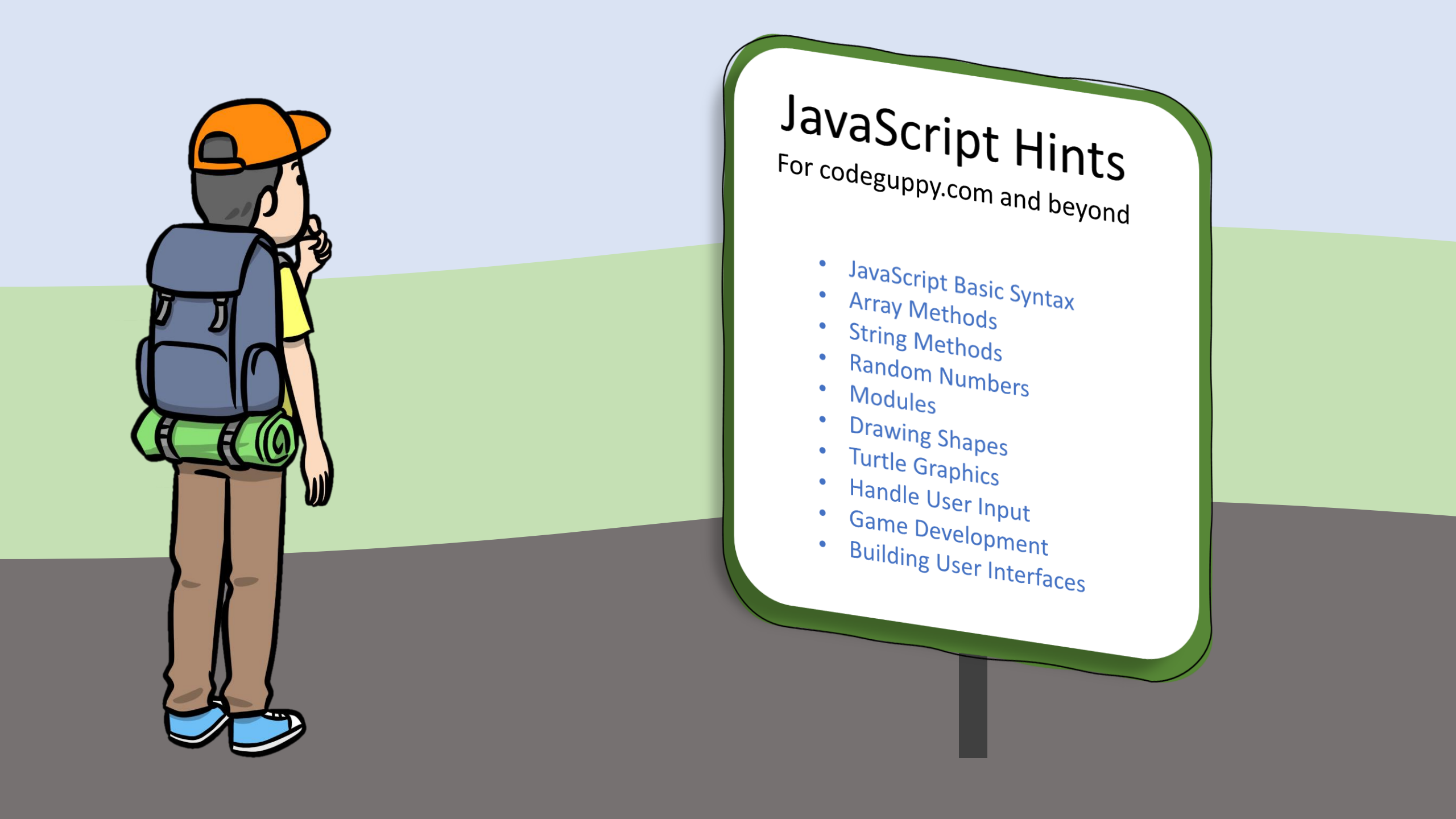

JavaScript Hints
For codeguppy.com and beyond
JavaScript Basic Syntax
Array Methods
String Methods
Random Numbers
Modules
Drawing Shapes
Turtle Graphics
Handle User Input
Game Development
Building User Interfaces

JavaScript Basic Syntax

JavaScript Basic Syntax: Variables

```
let x;
```
Declare variable x

```
let x = 1;
```
Declare x and initialize it with a numerical value

```
let s = "Hello, World!";
```
Declare s and initialize it with a string

```
x = 100;
```
Assign number 100 to variable x

```
s = "Hello";
```
Assign string "Hello" to variable s

```
ar = [];
```
Assign an empty array to variable ar

```
ar = [1, 2, 3];
```
Assign an array of 3 numbers to variable ar

```
ar = ["A", "B"];
```
Assign and array of 2 strings to variable ar

```
o = {   Type: 'car',
        x : 100,
        y : 200
    };
```
Assign an inline object to variable o

```
sum = a + b;
```
Variable sum is equal to a + b

```
avg = (a + b) / 2;
```
Assign an expression to variable avg

```
sum = sum + 10;
```
Variable sum is increased by 10 (the new sum becomes the older sum + 10)

```
i++;
```
Variable i is increased (incremented) by 1

```
i += 2;
```
Variable i is incremented by 2

JavaScript Basic Syntax: *if* statement

```
if (mouseX < width)
{

}
```

Executes the block of instructions between { } if condition is true

```
if (hour < 12)
{

}
else
{

}
```

Executes the first block of instructi ons if condition is true

… otherwise the second block

```
if (minute <= 15)
{
}
else if(minute <= 30)
{
}
else
{
}
```

If the first condition is *true*, then the first block will be executed and the others not.

If the first condition is not true, the *else if* is used to test another condition, and if is true, the block of that *else if* is executed.

The block after the last *else* is executed only if no other condition was *true* until that point.

JavaScript Basic Syntax: *for* loop

Prints numbers from 0 to 4 using a for loop and println

```
for(let i = 0; i < 5; i++)
{
    println(i);
}
```

0
1
2
3
4

Prints numbers from 10 down to 0 using a for loop

```
for(let i = 10; i >= 0; i--)
{
    println(i);
}
```

10
9
8
7
6
5
4
3
2
1
0

Prints even numbers from 0 to 100

```
for(let i = 0; i <= 100; i+=2)
{
    println(i);
}
```

0
2
4
...
100

Print all elements of an array

```
let ar = [10, 20, 30];

for(let element of ar)
{
    println(element);
}
```

10
20
30

JavaScript Basic Syntax: *while / do while*

Print numbers from 0 to 9 using a while loop

```
let i = 0;

while(i < 10)
{
    println(i);
    i++;
}
```

0
1
2
3
4
5
6
7
8
9

Print numbers from 0 to 10 using a do while loop

```
let i = 0;

do
{
    println(i);
    i++;
}
while(i < 10)
```

0
1
2
3
4
5
6
7
8
9

do while loop places condition after the code block, therefore the block can execute at least once, even if the condition is false.

JavaScript Basic Syntax: Functions

Defining and calling the function balloon

```
function balloon(x, y, shapeColor)
{
    let r = 30;
    let stringLen = 100;

    fill(shapeColor);
    stroke(shapeColor);

    circle(x, y, r);
    line(x, y + r, x, y + r + stringLen);
}
```

Function balloon draws a balloon using simple shapes such as circle and line

It expects as arguments the coordinates for balloon center and the color of the balloon

```
balloon(100, 100, "red");
balloon(300, 300, "blue");
balloon(500, 200, "yellow");
```

Call function balloon with different parameters

Functions that return values

```
function addNumbers(x, y)
{
    return x + y;
}

let sum = addNumbers(100, 200);
println(sum);
```

Call the function and store the value returned in a variable

Printing data

Print the numbers from 0 to 9

```
for(let i = 0; i < 10; i++)
{
    println(i);
}
```

`println` is great for quickly printing information while learning to code, testing or debugging programs… or practicing algorithms!

You can use `print` if you want to print on the same line.

Print the first 10 prime numbers

```
let found = 0;
let n = 0;

while(found < 10)
{
    if (isPrime(n))
    {
        println(n);
        found++;
    }

    n++;
}
```

```
// Returns true if specified number is prime

function isPrime(n)
{
    if (n < 2)
        return false;

    let max = sqrt(n);

    for(let i = 2; i <= max; i++)
    {
        if (n % i === 0)
            return false;
    }

    return true;
}
```

Array Methods

JavaScript Array Methods. Part 1

Declaring and initializing an empty array

```
let ar = [];
```

Declaring and initializing an array of 3 numbers

```
let ar = [10, 20, 30];
```

Length of an array

```
let ar = [10, 20, 30];
println(ar.length);
```

3

Append an element at the end of the array

```
let ar = [10, 20, 30];
ar.push(100);
println(ar);
```

[10, 20, 30, 100]

Insert an element at the beginning of an array

```
let ar = [10, 20, 30];
ar.unshift(1);
println(ar);
```

[1, 10, 20, 30]

Insert an element at an arbitrary position

```
let ar = [10, 20, 30];
ar.splice(1, 0, 15);
println(ar);
```

[10, 15, 20, 30]

After element with position 1, delete 0 elements, and insert number 15

JavaScript Array Methods. Part 2

Read the value of element 2 of an array

```
let ar = [10, 20, 30];
println(ar[2]);
```

30

Calculate the sum of elements of an array

```
let ar = [10, 20, 30];
let sum = 0;

for(let element of ar)
{
    sum += element;
}

println(sum);
```

Use a classic for loop to iterate the elements of the array and add them to a sum variable

Assign a different value to al element of an array

```
let ar = [10, 20, 30];
ar[2] = 100;
println(ar);
```

[10, 20, 100]

Access the first element

```
let ar = [10, 20, 30];
println(ar[0]);
```

Access the last element

```
let ar = [10, 20, 30];
let len = ar.length;
println(ar[len - 1]);
```

30

JavaScript Array Methods. Part 3

Remove the first element of the array

```
let ar = [10, 20, 30];
ar.shift();
println(ar);
```

[20, 30]

Remove the last element of the array

```
let ar = [10, 20, 30];
ar.pop();
println(ar);
```

[10, 20]

Remove an element at an arbitrary position

```
let ar = [10, 20, 30];

// 0 -> element index
// 1 -> number of elements to remove
ar.splice(0, 1);
println(ar);
```

[20, 30]

Remove all elements of an array

```
let ar = [10, 20, 30];

ar.length = 0;

println(ar);
```

[]

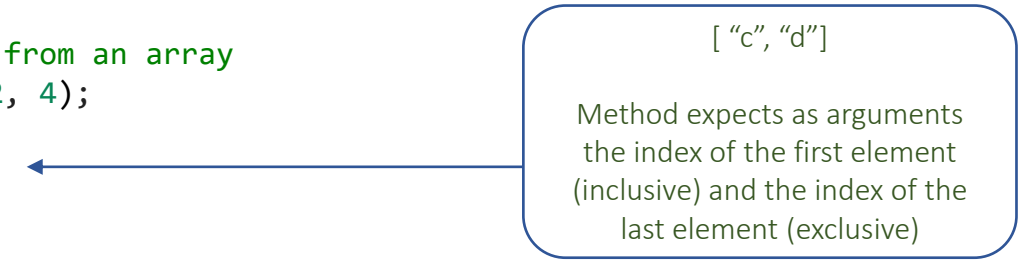
JavaScript Array Methods. Part 4

Concatenate two arrays

```
// Merge / concatenate 2 arrays
let ar1 = ["a", "b", "c"];
let ar2 = ["d", "e", "f"];

let ar = ar1.concat(ar2);

println(ar);
```

Extract a slice of an array

```
let ar = ["a", "b", "c", "d", "e", "f"];

// Extracting a 'slice' from an array
let arSlice = ar.slice(2, 4);

println(arSlice);
```



Joining elements of an array in a string

```
let ar = ["a", "b", "c", "d", "e", "f"];

// Join all elements in a string using separator ;
let s = ar.join(";");

println(s);
```

String Methods

JavaScript String Methods. Part 1

Length of a string

```
let txt = "JavaScript";
println(txt.length);
```

10

Iterating all characters of a string

```
let txt = "JavaScript";

for(let chr of txt)
{
    println(chr);
}
```

"J"
'a"
"v"
"a"
"S"
"c"
"r"
"i"
"p"
"t"

Accessing string characters by position

```
let txt = "JavaScript";

for(let i = 0; i < txt.length; i++)
{
    println(txt[i]);
}
```

Not recommended for Unicode strings

"J"
'a"
"v"
"a"
"S"
"c"
"r"
"i"
"p"
"t"

JavaScript String Methods. Part 2

Converting text to uppercase

```
let txt = "JavaScript";

txt = txt.toUpperCase();
println(txt);
```

"JAVASCRIPT"

Converting text to lowercase

```
let txt = "JavaScript";

txt = txt.toLowerCase();
println(txt);
```

"javascript"

Determine if the string contains another substring

```
let txt = "Coding is cool!";
let search = "cool";

if (txt.includes(search))
{
    println(search + " was found in " + txt);
}
```

true

JavaScript String Methods. Part 3

Determine if the string starts with a specified prefix

```
let txt = "JavaScript is cool!";
let search = "JavaScript";

if (txt.startsWith(search))
{
    println(txt + " starts with " + search);
}
```

true

Determine if the string ends with a specified suffix

```
let txt = "JavaScript is cool!";
let search = "!";

if (txt.endsWith(search))
{
    println("It is an exclamation!");
}
```

true

Find the position of a substring. Search starts at the beginning

```
let txt = "JavaScript is cool!";
let search = "cool";

let foundAt = txt.indexOf(search);

if (foundAt < 0)
    println("Not found!");

else
    println("Found at position " + foundAt);
```

14

JavaScript String Methods. Part 4

Find the position of a substring. Search starts at specified index.

```
let txt = "JavaScript is cool! Super cool!";

let search = "cool";
let startAt = 18;

let foundAt = txt.indexOf(search, startAt);

if (foundAt < 0)
    println("Not found!");

else
    println("Found at position " + foundAt);
```

26

Extract a substring from the string

```
let txt = "JavaScript is cool!";

let index1 = 14;
let index2 = 18;

let txt2 = txt.substring(index1, index2);

println(txt2);
```

“cool”

JavaScript String Methods. Part 5

Remove whitespaces from beginning and end of the string

```
let txt = "    I love coding !    ";

txt = txt.trim();
println("'" + txt + "'");
```

"I love coding !"

Remove whitespaces from beginning of the string

```
let txt = "    I love coding !    ";

txt = txt.trimStart();
println("'" + txt + "'");
```

"I love coding !   "

Remove whitespaces from the end of the string

```
let txt = "    I love coding !    ";

txt = txt.trimEnd();
println("'" + txt + "'");
```

"   I love coding !"

JavaScript String Methods. Part 6

Pads the start of the string with another string

```
let no = 3;

let txt = no.toString(2).padStart(8, '0');
println(txt);
```

"00000011"

Pads the end of the string with another string

```
let n1 = "1";
let n2 = "3";

txt = n1 + "." + n2.padEnd(4, '0');
println(txt);
```

"1.3000"

JavaScript String Methods. Part 7

Codes of characters

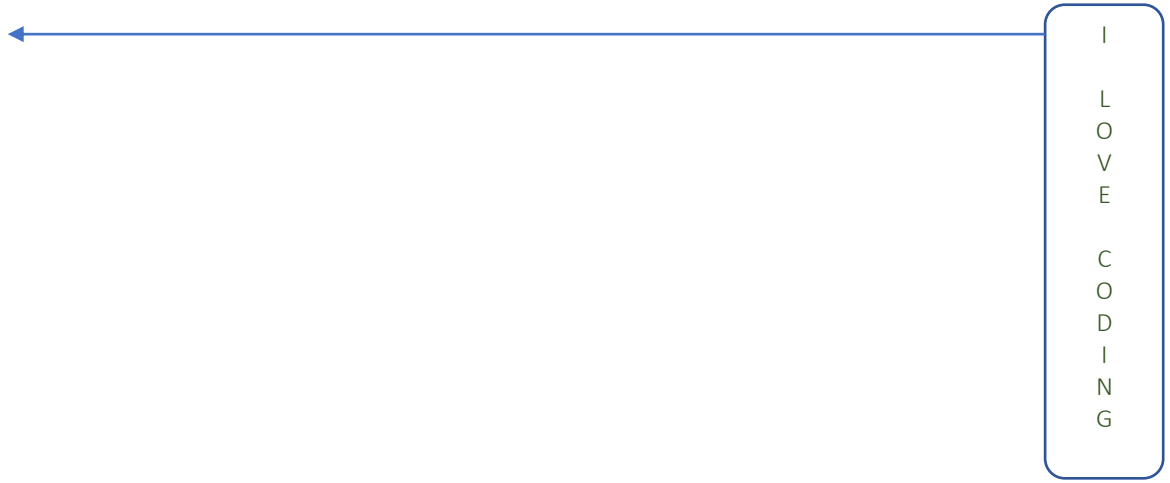
```
let txt = "JavaScript";

for(let chr of txt)
{
    // Obtain the Unicode code point value
    // ... identical to ASCII code for the range of ASCII values
    let code = chr.codePointAt(0);

    let line = chr + "\t" + code.toString() + "\t" +
                     code.toString(16).toUpperCase() + "\t" +
                     code.toString(2).padStart(7, "0");

    println(line);
}
```

J	74	4A	1001010
a	97	61	1100001
v	118	76	1110110
a	97	61	1100001
S	83	53	1010011
c	99	63	1100011
r	114	72	1110010
i	105	69	1101001
p	112	70	1110000
t	116	74	1110100

JavaScript String Methods. Part 8

Characters from codes

```
let msg = "73 32 76 79 86 69 32 67 79 68 73 78 71"
let base = 10;

let arMsg = msg.split(" ");

for(let i = 0; i < arMsg.length; i++)
{
    if (!arMsg[i])
        continue;

    let code = parseInt(arMsg[i], base);

    // Obtain the character from the Unicode code point
    // (the Unicode code point is the same with ASCII code for range of ASCII values)
    let chr = String.fromCodePoint(code);

    println(chr);
}
```



Random Numbers

Random Numbers. Part 1

Random floating-point number between 0 and 1 (1 not included)

```
// Same as Math.random()
let n = random();
println(n);
```

Random floating-point number between 0 and n (n not included)

```
let n = random(100);
println(n);
```

Random floating-point number between n1 and n2 (n2 not included)

```
let n = random(-100, 100);
println(n);
```

Random integer between min and max (both included)

```
// You can use either randomInt or randomNumber
let n = randomInt(0, 10);
println(n);
```

Random Numbers. Part 2

Random char between chr1 and chr2 (both included)

```
function randomChar(chr1, chr2)

let char = randomChar("A", "Z");
println(char);
```

Random element of an array

```
let ar = ["J", "a", "v", "a", "S", "c", "r", "i", "p", "t"];

let char = random(ar);
println(char);
```

Shuffle an array

```
let ar = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
let ar2 = ar.shuffle();

println(ar2);
```

Modules

Modules

Main Program

```
const math = require("MathUtils");

let sum = math.add(2, 3);
let prod = math.prod(3, 4);

println(sum);
println(prod);
```

MathUtils

```
function add(a, b)
{
    return a + b;
}

function prod(a, b)
{
    return a * b;
}
```

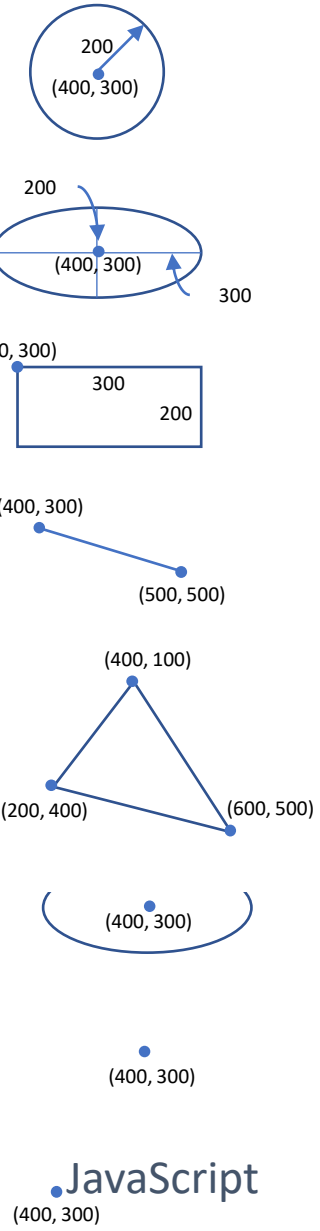
Modules provide encapsulation for code and data / variables

All functions from a module are automatically “exported”

Drawing Shapes

Drawing. Shapes

```
circle(400, 300, 200);

ellipse(400, 300, 300, 200);

rect(400, 300, 300, 200);

line(400, 300, 500, 500);

triangle(400, 100, 200, 400, 600, 500);

arc(400, 300, 300, 200, 0, 180);

point(400, 300);

text('JavaScript', 400, 300);
```



Drawing. Shape Settings

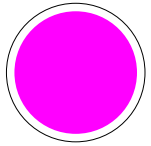
Set the size of text to 20

```
textSize(20);
text("JavaScript", 400, 300);
```

JavaScript

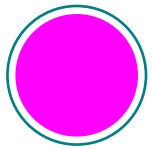
Set "Magenta" as the color to fill shapes

```
fill('Magenta');
circle(400, 300, 100);
```



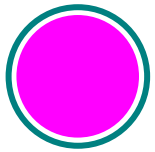
Set "Teal" as the color to draw shapes

```
stroke('Teal');
circle(400, 300, 100);
```



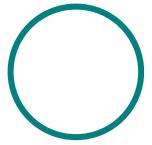
Set the line thickness to 2 px

```
strokeWeight(2);
circle(400, 300, 100);
```



Draw empty shapes, without fill color

```
noFill();
circle(400, 300, 100);
```



Draw shapes without an outline

```
noStroke();
fill("lightblue");
circle(400, 300, 100);
```

Drawing. Colors

Set the color for drawing	Set the color for filling	Specifying color
noStroke() stroke(color) strokeWeight(weight)	background(color) fill(color) noFill()	- #RRGGBB (red, green, blue components) - Predefined color

Popular colors

IndianRed, LightCoral, Salmon, DarkSalmon, LightSalmon, Crimson, Red, FireBrick, DarkRed

Pink, LightPink, HotPink, DeepPink, MediumVioletRed, PaleVioletRed

LightSalmon, Coral, Tomato, OrangeRed, DarkOrange, Orange

Gold, Yellow, LightYellow, LemonChiffon, LightGoldenrodYellow, PapayaWhip, Moccasin, PeachPuff, PaleGoldenrod, Khaki, DarkKhaki

Lavender, Thistle, Plum, Violet, Orchid, Fuchsia, Magenta, MediumOrchid, MediumPurple, RebeccaPurple, BlueViolet, DarkViolet, Dark Orchid, DarkMagenta, Purple, Indigo, SlateBlue, DarkSlateBlue, MediumSlateBlue

GreenYellow, Chartreuse, LawnGreen, Lime, LimeGreen, PaleGreen, LightGreen, MediumSpringGreen, SpringGreen, MediumSeaGreen, SeaGreen, ForestGreen, Green, DarkGreen, YellowGreen, OliveDrab, Olive, DarkOliveGreen, MediumAquamarine, DarkSeaGreen, Light SeaGreen, DarkCyan, Teal

Aqua, Cyan, LightCyan, PaleTurquoise, Aquamarine, Turquoise, MediumTurquoise, DarkTurquoise, CadetBlue, SteelBlue, LightSteelBlue , PowderBlue, LightBlue, SkyBlue, LightSkyBlue, DeepSkyBlue, DodgerBlue, CornflowerBlue, MediumSlateBlue, RoyalBlue, Blue, Mediu mBlue, DarkBlue, Navy MidnightBlue

Cornsilk, BlanchedAlmond, Bisque, NavajoWhite, Wheat, BurlyWood, Tan, RosyBrown, SandyBrown, Goldenrod, DarkGoldenrod, Peru, Chocolate, SaddleBrown, Sienna, Brown, Maroon

White, Snow, HoneyDew, MintCream, Azure, AliceBlue, GhostWhite, WhiteSmoke, SeaShell, Beige, OldLace, FloralWhite, Ivory, Antiqu eWhite, Lin n, LavenderBlush, MistyRose

Gainsboro, LightGray, Silver, DarkGray, Gray, DimGray, LightSlateGray, SlateGray, DarkSlateGray, Black

Animations. Bouncing ball

```
let x = 400;
let y = 300;

let dx = 1;
let dy = 1;

let speed = 3;

function loop()
{
    clear();

    circle(x, y, 10);

    x += speed * dx;
    y += speed * dy;

    if (x < 0 || x > width)
        dx *= -1;

    if (y < 0 || y > height)
        dy *= -1;
}
```

Holds animation state.

loop is a special function. If defined, will be automatically invoked up to 60 times / second

Clear the frame

Draw the objects in the frame (in this case just the ball)

Update state. The next frame that will be displayed will use these new values.

Turtle Graphics

Turtle Graphics. Part 1

```
home();
```
Reset the default turtle to home position

```
pencolor("Red");
```
Sets to Red the pen color of the default turtle

```
pensize(2);
```
Sets to 2 the pen size of the default turtle

```
pendown();
```
Put the pen on the paper. The turtle will draw

```
penup();
```
Raise the pen from the paper. The turtle will advance but not draw

```
setposition(100, 100);
```
Move the turtle to an arbitrary position on the canvas

```
left(30);
```
Turns the default turtle to the left by the number of specified degrees

```
right(30);
```
Turns the default turtle to the right by 30 degrees

Turtle Graphics. Part 2

```
setheading(180);
```

Sets the turtle heading (direction) to an arbitrary angle

```
forward(100);
```

Moves the turtle forward by number of specified pixels.
The turtle moves in the direction that was previously set with *left*, *right* or *setheading*. If the pen is on the paper, the turtle will draw.

```
back(100);
```

The turtle moves in the opposite direction than would move with forward

```
let p = position();
println(p[0]);
println(p[1]);
```

Retrieve the x and y position of the default turtle as an array of 2 numbers

```
let angle = heading();
println(angle);
```

Retrieve the default turtle direction in degrees

```
let t1 = createTurtle();
let t2 = createTurtle();
t1.pencolor("Red");
t2.pencolor("Blue");
```

Working with multiple turtles

```
let t = getDefaultTurtle();
t.forward(100);
```

Get the default turtle

Handle User Input

Handle User Input. Keyboard Events

```
function keyPressed()
{
    clear();
    text(key, 400, 300);
    text(keyCode, 400, 320);
}
```

keyPressed event. Executes once when a key is pressed

```
function keyReleased()
{
    clear();
    text(key, 400, 300);
    text(keyCode, 400, 320);
}
```

keyReleased event. Executes when a key is released

```
function keyTyped()
{
    clear();
    text(key, 400, 300);
    text(keyCode, 400, 320);
}
```

keyTyped event. Executes when a key is typed execept for special keys

Handle User Input. Mouse Events 1

```
function mouseClicked()
{
    circle(mouseX, mouseY, 10);
}
```

mouseClicked event. Executes once when the mouse is pressed and released

```
function mousePressed()
{
    stroke("red");
    circle(mouseX, mouseY, 10);
}
```

mousePressed event. Executes once when the mouse button is pressed

```
function mouseReleased()
{
    stroke("blue");
    circle(mouseX, mouseY, 10);
}
```

mouseReleased event. Executes when the mouse button is released

```
function doubleClicked()
{
    circle(mouseX, mouseY, 10);
}
```

doubleClicked event. Executes when the mouse is double clicked

Handle User Input. Mouse Events 2

```
function mouseMoved()
{
    line(mouseX, mouseY,
                 pmouseX, pmouseY);
}
```

mouseMoved event. Executes when the mouse is moved and button is not pressed

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

mouseDragged event. Executes when the mouse is moved and a button is pressed

```
function mouseWheel()
{

}
```

mouseWheel event. Executes when the user uses the mouse wheel or touchpad

Handle User Input. Keyboard System Variables

```
noStroke();
text("Press any key to change color", 10, 10);

function loop()
{
    let color = keyIsPressed ? "Red" : "Green";

    clear();
    fill(color);
    circle(400, 300, 100);
}
```

Boolean system variable that indicates if a key is pressed.

```
function keyPressed()
{
    if (key.toLowerCase() === "s")
    {
        showScene("Game");
    }
}
```

System variable containing the last typed key.

```
function keyPressed()
{
    let ENTER_KEYCODE = 13;

    if (keyCode === ENTER_KEYCODE)
    {
        showScene("Game");
    }
}
```

System variable containing the code of the last key pressed.

The following constants can be used instead of a numeric key code: LEFT_ARROW, RIGHT_ARROW, UP_ARROW, DOWN_ARROW.

Handle User Input. Mouse System Variables

```
function loop()
{
    let drawColor = mouseButton === LEFT
                    ? "Red" : "Blue";

    stroke(drawColor);

    if (mouseIsPressed)
        line(mouseX, mouseY, pmouseX, pmouseY);
}
```

System variable containing the pressed mouse button. It has one of these values LEFT, RIGHT, CENTER.

Boolean system variable indicating if any mouse button is pressed

Horizontal coordinate of the mouse cursor.

Vertical coordinate of the mouse cursor.

Previous horizontal coordinate of the mouse cursor

Previous vertical coordinate of the mouse cursor.

User Input. keyIsDown() / keyWentDown() functions

```
let shipX = width / 2;
let fireLaser = false;

function loop()
{
    if (keyIsDown(LEFT_ARROW))
        shipX -= 10;

    else if (keyIsDown(RIGHT_ARROW))
        shipX += 10;

    fireLaser = false;

    if (keyWentDown(32)) // SPACE key
        fireLaser = true;

    draw();
}

function draw()
{
    clear();

    fill("Magenta");
    rect(shipX, height - 40, 100, 20);

    if (fireLaser)
    {
        stroke("Red");
        line(shipX + 50, height - 40, shipX + 50, 0);
    }
}
```

Use *keyIsDown()* function inside the loop() event to detect if the specified key is pressed. You need to specify the key code.

The following constants can be used instead of a numeric key code: LEFT_ARROW, RIGHT_ARROW, UP_ARROW, DOWN_ARROW.

keyWentDown() is similar to *keyIsDown()* but returns true just once per key pressed.

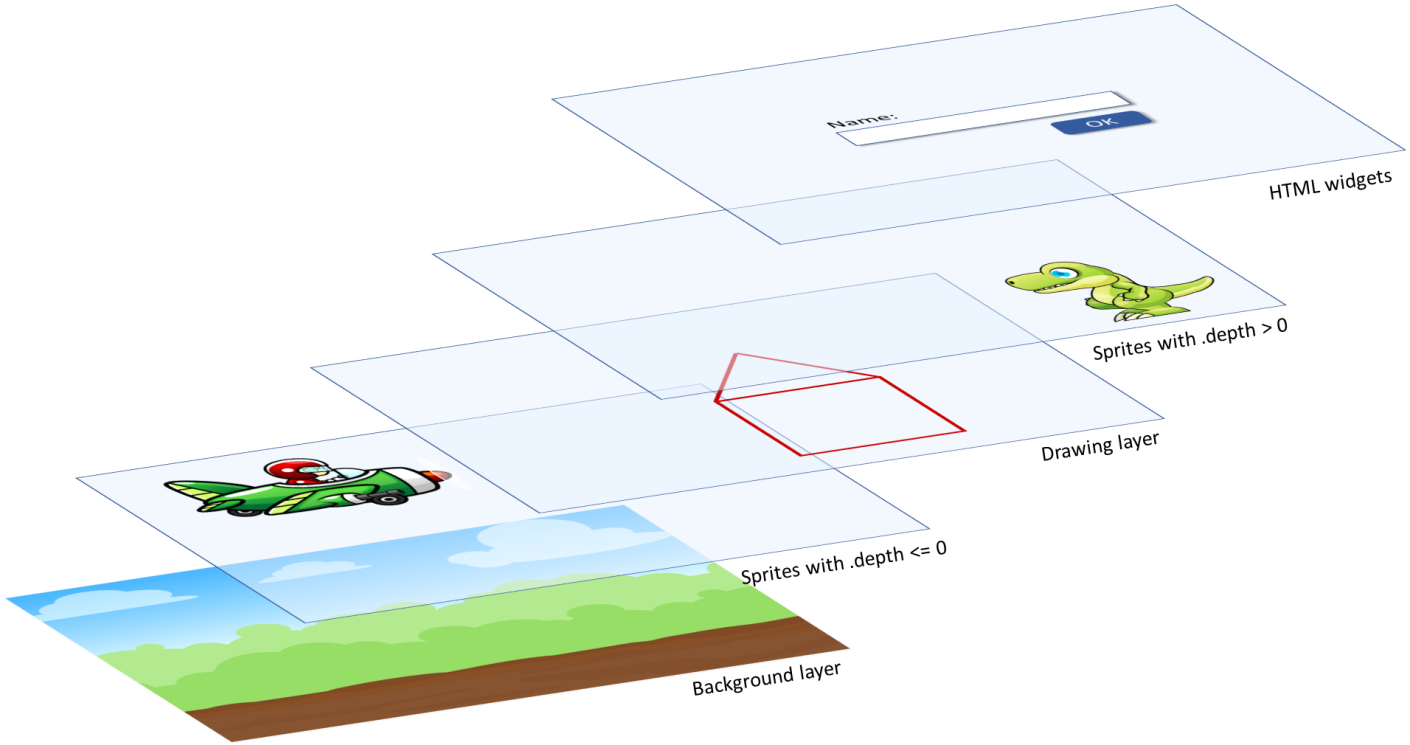
To retrigger the function, the user need to release the key and press it again.

Game Development

Game Development. Layers

codeguppy.com has a layered drawing architecture. There are up to 5 drawing layers on top of the canvas at any time as shown in the following diagram.

The engine combines automatically all the layers and displays the final image on the screen.



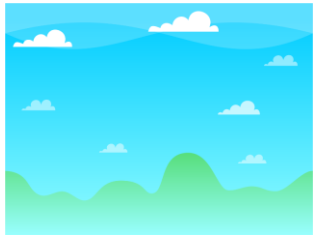
Game Development. Background

```
background('LightBlue’);
```

Use a popular named color

```
background('#008080’);
```

Use an RGB color



```
background('Summer');
```

Use an image from library

Game Development. Loading Built-in Sprites

```
sprite('plane’);
```

Load and show sprite “plane”

```
sprite('plane', 400, 200);
```

Load and show sprite “plane” at coordinates (400, 200)

```
sprite('plane', 0.5);
```

Load and show sprite “plane” using a 0.5 size scaling factor

```
sprite('plane', 400, 150, 0.5);
```

Load and show sprite “plane” at coordinates and scale specified

```
sprite('plane.shoot', 400, 150, 0.5);
```

Load and show the animation “shoot” of sprite “plane”.

Display is done at coordinates and scale specified.

Game Development. Loading Custom Sprites

sprite receives an image object as parameter.

img converts text to an Image object.

```
sprite(img`
    . . . . . . . . . . . . . . . .
    . . . . . . 4 4 5 . . . . . . .
    . 8 8 8 8 8 4 4 4 8 8 8 8 8 . .
    . . . . . . . f . . . . . . . .
    . . . . . 8 8 8 8 8 . . . . . .
    . . . . 8 1 1 8 1 1 8 . . . . .
    . . . . 8 1 f 8 f 1 8 . . . . .
    . . 8 . 8 8 8 2 8 8 8 . 8 . . .
    . . 8 8 8 2 8 8 8 2 8 8 8 . . .
    . . . . 8 8 2 2 2 8 8 . . . . .
    . . . . 8 8 8 8 8 8 8 . . . . .
    . . . . 8 8 8 8 8 8 8 . . . . .
    . . . . 8 . . . . . 8 . . . . .
    . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . .
`, 10);
```

16 color image in Microsoft MakeCode Arcade format

- 0 - (transparent)
- 1 - #ffffff
- 2 - #ff2121
- 3 - #ff93c4
- 4 - #ff8135
- 5 - #fff609
- 6 - #249ca3
- 7 - #78dc52
- 8 - #003fad
- 9 - #87f2ff
- a - #8e2ec4
- b - #a4839f
- c - #5c406c
- d - #e5cdc4
- e - #91463d
- f - #000000

Scaling factor

Game Development. Animated Custom Sprites

Define animation frames

```
// Note: Shark images are from Microsoft MakeCode Arcade
// https://arcade.makecode.com/

let shark1 = img`
    . . . . . . . . . . . . . c c f f f . . . . . . . . . . . . . .
    . . . . . . . . . . . . c c d d b c f . . . . . . . . . . . . . .
    . . . . . . . . . . . c c d d b b f . . . . . . . . . . . . . . .
    . . . . . . . . . . . f c c b b c f . . . . . . . . . . . . . . .
    . . . . . f f f f f c c c c c c f f . . . . . . . . . c c c . . .
    . . . f f b b b b b b b b c b b b b c f f f . . . . c c b b c . . .
    . . f b b b b b b b b b c b c b b b b c c c f f . c d b b c . . . .
    f f b b b b b b f f b b c b c b b b c c c c c f c d b b f . . . .
    f b c b b b 1 1 f f 1 b c b b b b b c c c c c f f b b f . . . . .
    f b b b 1 1 1 1 1 1 1 1 b b b b b c c c c c c c b b c f . . . . .
    . f b 1 1 1 3 3 c c 1 1 b b b b c c c c c c c c c c c f . . . . .
    . . f c c c 3 1 c 1 1 1 b b b c c c c c b d b f f b b c f . . . .
    . . . f c 1 3 c 1 1 1 c b b b f c d d d d c c . . f b b f . . . .
    . . . . f c c c 1 1 1 f b d b b c c d c c . . . . . f b b f . . .
    . . . . . . . . c c c c f c d b b c c . . . . . . . . f f f . . .
    . . . . . . . . . . . . . f f f f f . . . . . . . . . . . . . . .
`;

let shark2 = img`...`;

let shark3 = img`...`;

let shark4 = img`...`;

sprite([shark1, shark2, shark3, shark4], 400, 300, 2);
```

Array of images for animation

Game Development. Custom Sprite with multiple animations

Define frames for all the animations that make up the custom sprite

```
let ship1 = img`
    . . . . . . . . . . . . . . . .
    8 8 1 . . . . . . . . . . . . .
    2 2 2 2 . . . . . . . . . . . .
    2 2 2 2 . . 9 9 9 9 . . . . . .
    8 8 8 8 8 9 9 9 9 9 9 . . . . .
    8 8 8 8 8 9 9 9 9 9 9 9 . . . .
    2 2 2 2 2 9 9 9 9 9 9 9 2 . . .
    2 2 2 2 2 2 2 2 2 2 2 2 2 2 . .
    . 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    4 4 4 2 2 2 2 2 2 2 2 2 2 2 2 2
    4 4 4 4 2 2 8 8 8 8 8 8 8 2 2 .
    4 4 . . . 8 8 8 8 8 8 8 . . . .
    . . . . 8 8 8 8 8 8 8 . . . . .
    . . . 8 8 8 8 8 8 8 . . . . . .
    . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . .
`;

let ship2 = img`...`;

let shipLand1 = img`...`;

let shipLand2 = img`...`;

let oShip = {
    Flying : [ship1, ship2],
    LandingDown : [shipLand1],
    LandingUp : [shipLand2]
};

sprite(oShip, 40, 100, 3);
```

Object describing the custom sprite

Game Development. Basic sprite manipulation

```
let player = sprite('adventure_girl.idle', 400, 300, 0.5);
player.x = 100;
player.y = 100;
```

Reference to the sprite object

Change sprite position on the canvas

```
let plane = sprite('plane.fly', 0, 100, 0.5);
plane.velocity.x = 1;

plane.mirrorX(-1);
plane.rotation = 30
```

Move sprite automatically: Instruct the engine to automatically increase the .x coordinate 1 pixel at a time (per frame)

Flip the sprite on the X axis… and also rotate it 30 degrees

```
for(let i = 0; i < 10; i++)
{
    let flower = sprite(img`
        . . . . . . . . . . . . . . . .
        . . . . . . . 5 5 . . . . . . .
        . . . . 5 5 . 5 5 . 5 5 . . . .
        . . . . 5 5 5 5 5 5 5 5 . . . .
        . . 5 5 . 5 f e f e 5 . 5 5 . .
        . . 5 5 5 f e f e f e 5 5 5 . .
        . . . 5 f e f e f e f e 5 . . .
        . 5 5 5 e f e f e f e f 5 5 5 .
    `, random(width), random(-height, 0), 3);

    flower.velocity.y = random(1, 3);

    flower.rotationSpeed = 2;
}
```

Move and rotates sprites automatically. Instruct the engine to increase the .y coordinate with a number between 1 and 3, and also rotate the sprite 2 degrees at a time (per frame).

Game Development. Sprite Groups

```
let player = sprite('game.happy', 400, 300, 0.5);
let coins = new Group();

for(let i = 0; i < 10; i++)
{
    let coin = sprite('coin',
            random(100, 700), random(50, 550), 0.5);

    // add coin to the group
    coins.add(coin);
}

function loop()
{
    player.x = mouseX;
    player.y = mouseY;

    // check collision against the group
    player.collide(coins, onCollision)
}

function onCollision(player, coin)
{
    // remove coin from the group
    coins.remove(coin);

    coin.velocity.y = -10;
    coin.life = 100;
}
```

Create a player sprite

Create a new sprite group

Create 10 sprite coins

… and add all of them to the “coins” group.

Move the player sprite at the mouse coordinates.

Check if the player sprite is colliding with any coin in the “coins” group, and if yes, invoke the “onCollision” function

Upon collision, remove the coin from the group and make it “fly” outside of the screen, 10 pixels at a time … for 100 frames.

Game Development. Music and Sound Effects

```
music('Fun Background', 0.1);
```

Music file from the built-in library

Volume of music

Plays a music file in a loop at the specified volume. A new invocation of *music* instruction will start the new music file.

```
sound('zap1');
```

Plays a sound effect from the built-in library.

Multiple invocations of sound instruction, will mix and play the sounds in parallel.

Game Development. The Game Loop

If the game is using only sprites

```
background('Road');

let p = sprite('adventure_girl.idle', 400, 400, 0.5);

function loop()
{
    p.show("idle");

    if (keyIsDown(LEFT_ARROW))
    {
        p.mirrorX(-1);
        p.x -= 10;
        p.show("run");
    }
    else if (keyIsDown(RIGHT_ARROW))
    {
        p.mirrorX(1);
        p.x += 10;
        p.show("run");
    }
}
```

Inside the game loop, read the user input (keyboard / mouse) and then update the game state and sprites properties.

Sprites will automatically redraw according to thew newly set properties.

If the game is using shapes

```
background('Field');
textSize(40);

let plane = sprite('plane.fly', 50, 100, 0.3);
let textX = -280;

function loop()
{
    textX++;
    displayBanner();

    plane.x++;
}

function displayBanner()
{
    clear();
    fill("White");
    rect(textX - 10, 80, 250, 50);
    fill("Black");
    text("Hello, World!", textX, 120);
}
```

If your game is also using shapes, the easiest solution is to clear the screen between each frame, then redraw the shapes using the new game state.

Game Development. Multi-Scene Games

Code of first scene (e.g. Game Scene)

```
let data = {
    score : 1000,
    time : 10,
    bonusPoints : 100
}

...

if (won)
    showScene("Congrats", data);
```

Use `showScene` to transition to a new scene.

You can also pass optional arguments to the new scene.

Code of second scene (e.g. Congrats Scene)

```
background("Teal");

function enter()
{
    let data = sceneArgs;

    text("Score: " + data.score, 400, 300);
    text("Time: " + data.time, 400, 320);
    text("Bonus Points: " + data.bonusPoints, 400, 340);
}
```

Function `enter` is automatically executed each time the scene is shown.

Use `sceneArgs` to retrieve the arguments passed to the scene by the `showScene` function.

Game Development. Collisions between arbitrary shapes. Part 1

Detect collision between point and circle

```
let circleX = 400;
let circleY = 300;
let circleR = 200;

function loop()
{
    clear();

    let collide = collisionPointCircle(mouseX, mouseY, circleX, circleY, circleR);
    stroke(collide ? "red" : "black");

    circle(circleX, circleY, circleR);
}
```

`collisionPointCircle` receives as arguments the attributes of the point and the circle

Detect collision between point and line

```
let lineX1 = 300;
let lineY1 = 400;
let lineX2 = 500;
let lineY2 = 200;

function loop()
{
    clear();

    let collide = collisionPointLine(mouseX, mouseY, lineX1, lineY1, lineX2, lineY2);
    stroke(collide ? "red" : "black");

    line(lineX1, lineY1, lineX2, lineY2);
}
```

`collisionPointLine` receives as arguments the attributes of the point and the line

Game Development. Collisions between arbitrary shapes. Part 2

Detect collision between a point and a rectangle

```
let rectX = 250;
let rectY = 200;
let rectWidth = 300;
let rectHeight = 200;

function loop()
{
    clear();

    let collide = collisionPointRect(mouseX, mouseY, rectX, rectY, rectWidth, rectHeight);
    stroke(collide ? "red" : "black");

    rect(rectX, rectY, rectWidth, rectHeight);
}
```

`collisionPointRect` receives as arguments the attributes of the point and the rectangle

Detect collision between two circles

```
let circle1R = 50;
let circle2X = 400;
let circle2Y = 300;
let circle2R = 100;

function loop()
{
    clear();

    let circle1X = mouseX;
    let circle1Y = mouseY;

    let collide = collisionCircleCircle(circle1X, circle1Y, circle1R, circle2X, circle2Y, circle2R)
    stroke(collide ? "red" : "black");

    circle(circle1X, circle1Y, circle1R);
    circle(circle2X, circle2Y, circle2R);
}
```

`collisionCircleCircle` receives as arguments the attributes of the two circles

Game Development. Collisions between arbitrary shapes. Part 3

Detect collision between a circle and a rectangle

```
let circleR = 50;
let rectX = 250, rectY = 200, rectWidth = 300, rectHeight = 200;

function loop()
{
    clear();

    let circleX = mouseX;
    let circleY = mouseY;
    let collide = collisionCircleRect(circleX, circleY, circleR, rectX, rectY, rectWidth, rectHeight)
    stroke(collide ? "red" : "black");

    circle(circleX, circleY, circleR);
    rect(rectX, rectY, rectWidth, rectHeight);
}
```

`collisionCircleRect` receives as arguments the attributes of the circle and the rectangle

Detect collision between two rectangles

```
let rect1X = 0, rect1Y = 0, rect1Width = 100, rect1Height = 50;
let rect2X = 250, rect2Y = 200, rect2Width = 300, rect2Height = 200;

function loop()
{
    clear();

    rect1X = mouseX;
    rect1Y = mouseY;

    let collide = collisionRectRect(rect1X, rect1Y, rect1Width, rect1Height,
                                    rect2X, rect2Y, rect2Width, rect2Height);
    stroke(collide ? "red" : "black");

    rect(rect1X, rect1Y, rect1Width, rect1Height);
    rect(rect2X, rect2Y, rect2Width, rect2Height);
}
```

`collisionRectRect` receives as arguments the attributes of the two rectangles

Game Development. Collisions between arbitrary shapes. Part 4

Detect collision between two lines

```
let x1 = 400;
let y1 = 300;
let x2 = 0;
let y2 = 0;

let x3 = 300;
let y3 = 400;
let x4 = 500;
let y4 = 200;

function loop()
{
    clear();

    x2 = mouseX;
    y2 = mouseY;

    let collide = collisionLineLine(x1, y1, x2, y2, x3, y3, x4, y4);
    stroke(collide ? "Red" : "Black");

    line(x1, y1, x2, y2);
    line(x3, y3, x4, y4);
}

function mouseClicked()
{
    x1 = mouseX;
    y1 = mouseY;
}
```

`collisionLineLine` receives as arguments the attributes of the two lines

Game Development. Collisions between arbitrary shapes. Part 5

Detect collision between a line and a rectangle

```
let x1 = 400;
let y1 = 300;

let x3 = 350;
let y3 = 250;
let w = 300;
let h = 100;

function loop()
{
    clear();

    let x2 = mouseX;
    let y2 = mouseY;

    let v = collisionLineRect(x1, y1, x2, y2, x3, y3, w, h);
    stroke(v ? "Red" : "Black");

    line(x1, y1, x2, y2);
    rect(x3, y3, w, h);
}

function mouseClicked()
{
    x1 = mouseX;
    y1 = mouseY;
}
```

`collisionLineRect` receives as arguments the attributes of the line and rectangle

Building User Interfaces

Building User Interfaces. Edit Boxes and Buttons

```
text("Your name", 300, 90);
let nameBox = createEdit(300, 100, 200);
nameBox.onchange = handleNameChange;

text("Comments", 300, 190);
let commentsBox = createEdit(300, 200, 300, 100);
```

`createEdit` creates an input box at coordinates (300, 200) and width of 300 pixels. The last parameter, the height, is optional. If missing a single-line input will be created.

`createEdit` is returning a reference to the edit box object. You can use the following properties to manipulate edit box content.

.text	.width
.readonly	.height
.visible	.onchange

```
let btn = createButton(505, 100, 60, 20);
btn.text = "Enter";
btn.onclick = handleButtonClick;
```

`createButton` creates a button at specified coordinates and size.

`createButton` is returning a reference to the button object. You can use the following properties to manipulate button.

.text	.width
.visible	.height
.disabled	.onclick

```
function handleNameChange()
{
    commentsBox.text = "Your name is " + nameBox.text;
}


function handleButtonClick(sender)
{
    commentsBox.text += "\nThe name you typed is " + nameBox.text + "\n";
}
```

Visit https://codeguppy.com for more fun projects!

For news and updates follow @codeguppy on Twitter!