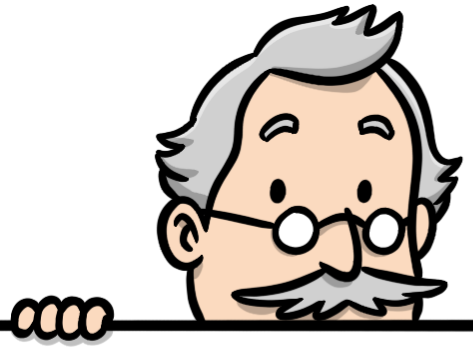
Introduction to coding

using JavaScript and codeguppy.com



Agenda

Part I: Introduction

- About codeguppy.com
- Creating accounts in codeguppy.com
- Environment overview

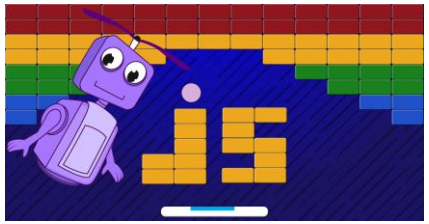
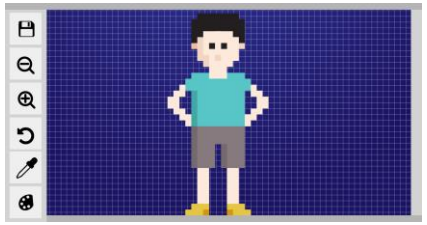
Part II: Steps into coding

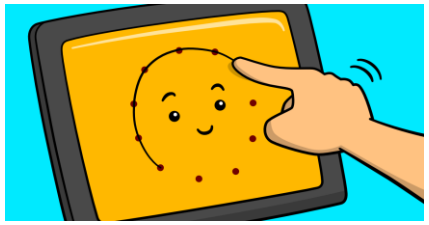
- Drag and drop coding
- Drawing with code
- Animations

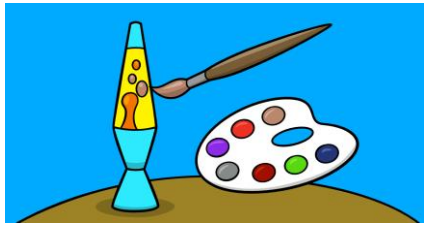
About codeguppy.com

- Online coding platform for kids, teens and code newbies
- Text-based coding
- JavaScript language
- Based on p5.js (Processing API)
- Code animations, games and other fun programs
- Built-in assets (backgrounds, sprites, etc.)
- Tons of projects. Browse – Remix – Share
- Free!

codeguppy.com



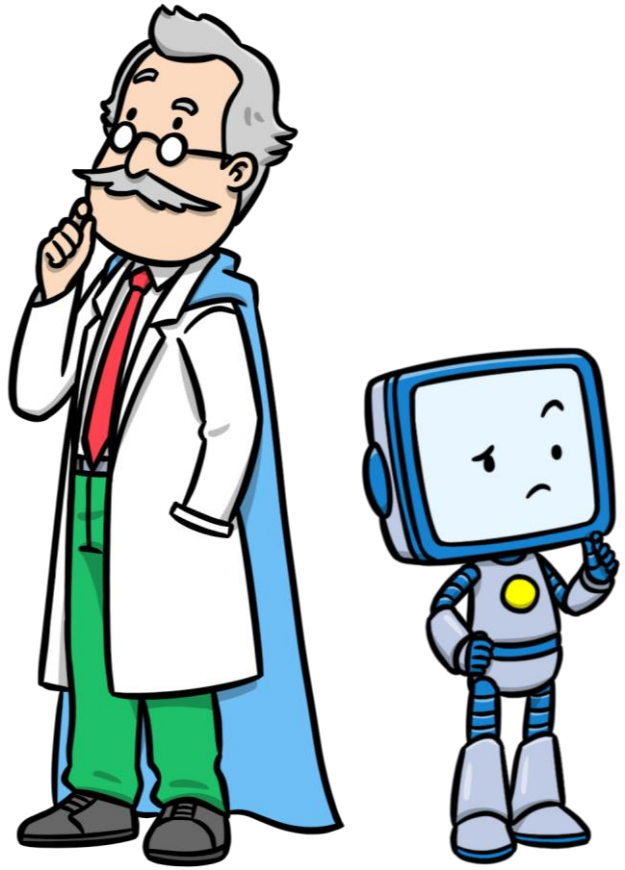


Why learn JavaScript?

- **JavaScript is a super-popular real-world programming language** used both by beginners but also by professional software developers. (The entire codeguppy.com was coded in JavaScript)

- People use JavaScript to build websites, online games and do creative coding. On codeguppy.com, you can use JavaScript to **draw with code**, implement **animations**, **build games** and other fun projects.

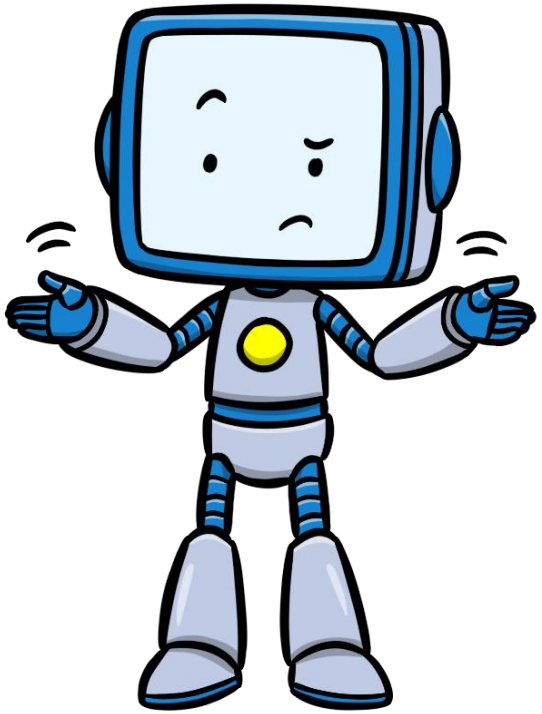
- Coding in a **text-based language** such as JavaScript has also other benefits beyond the world of programming. It enhances problem-solving skills but also attention to details.



Programming is like writing a book...

… except if you miss out a single comma on page 349 the whole thing you wrote makes no sense.

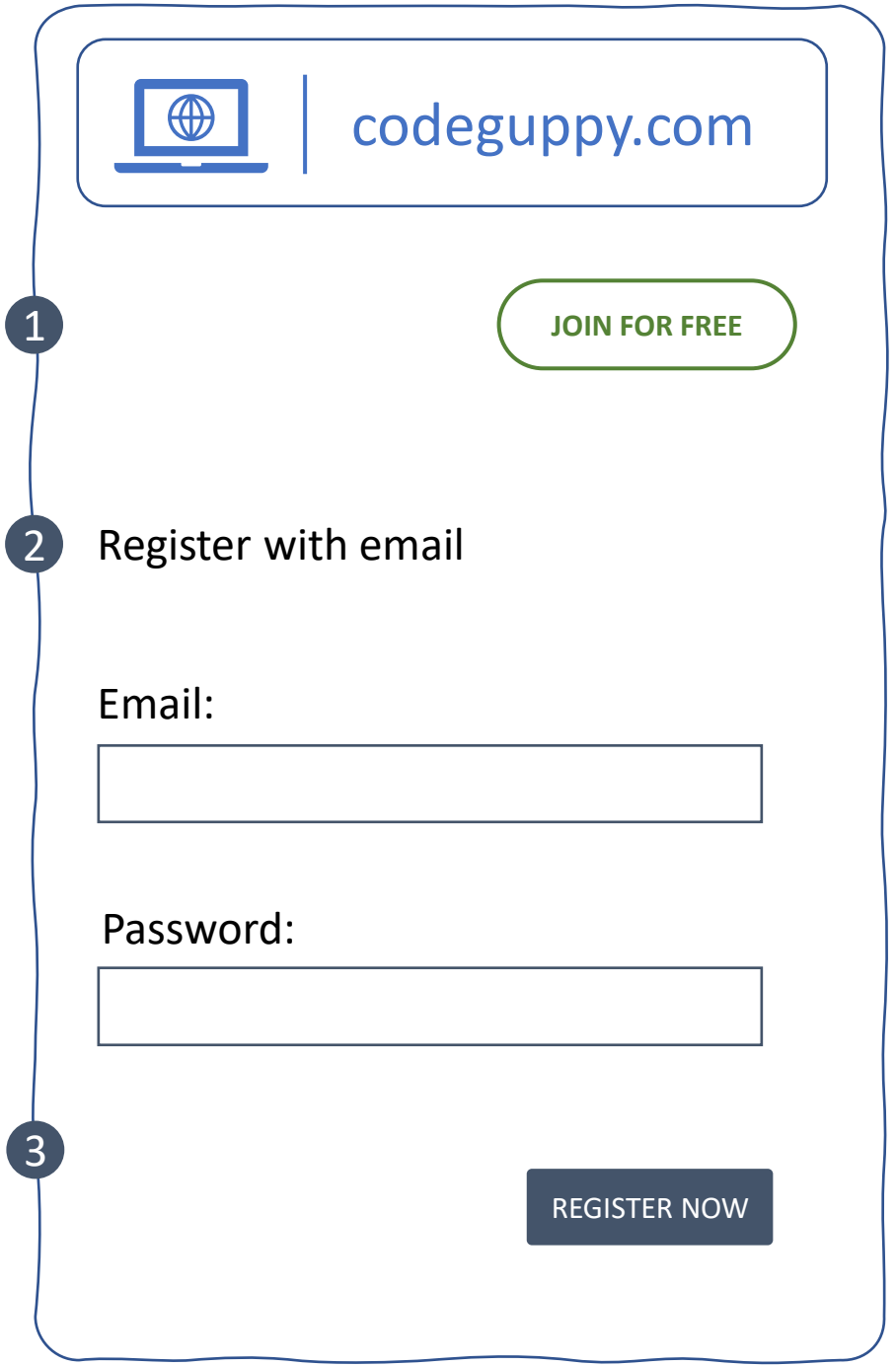
- Anonymous



Let’s open an account with codeguppy.com

- codeguppy.com gives you unlimited space in the cloud to write and store tons of programs
- But first you need to open an account with codeguppy.com
- You need to provide a valid email address and a password (website doesn’t ask any other personal details)

Note: After registration, you should get an email from codeguppy.com. Please verify your email by clicking the link in the email.

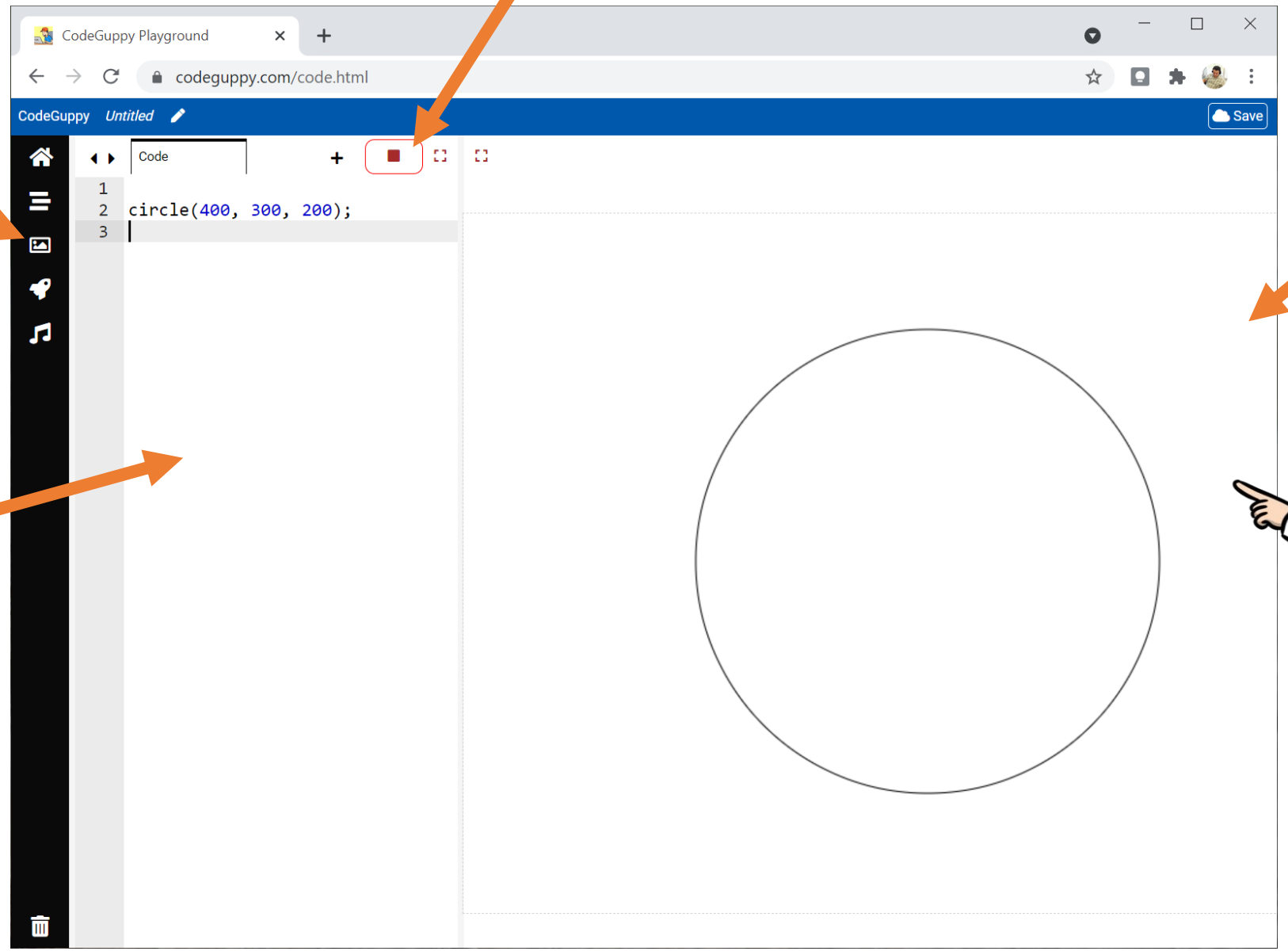


Run / Stop Button

Run or Stop your program.

Assets Toolbar

Browse sprite library, colors, etc.



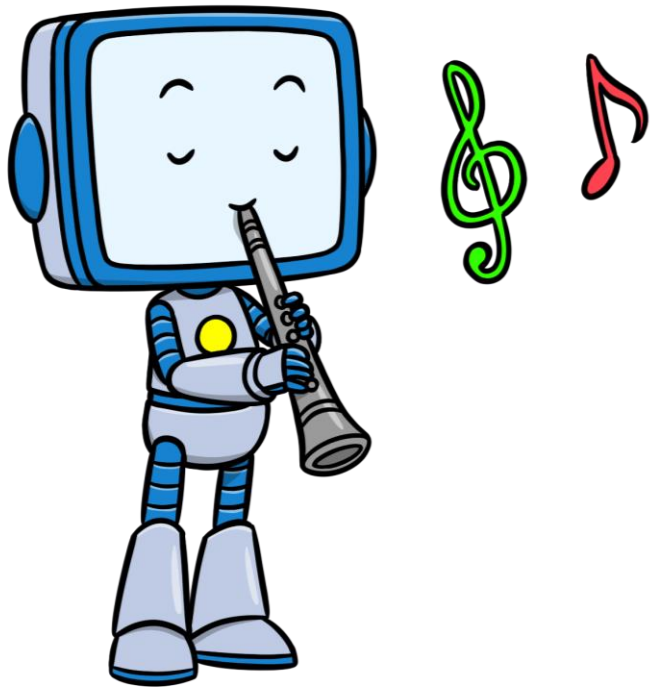
Output Canvas

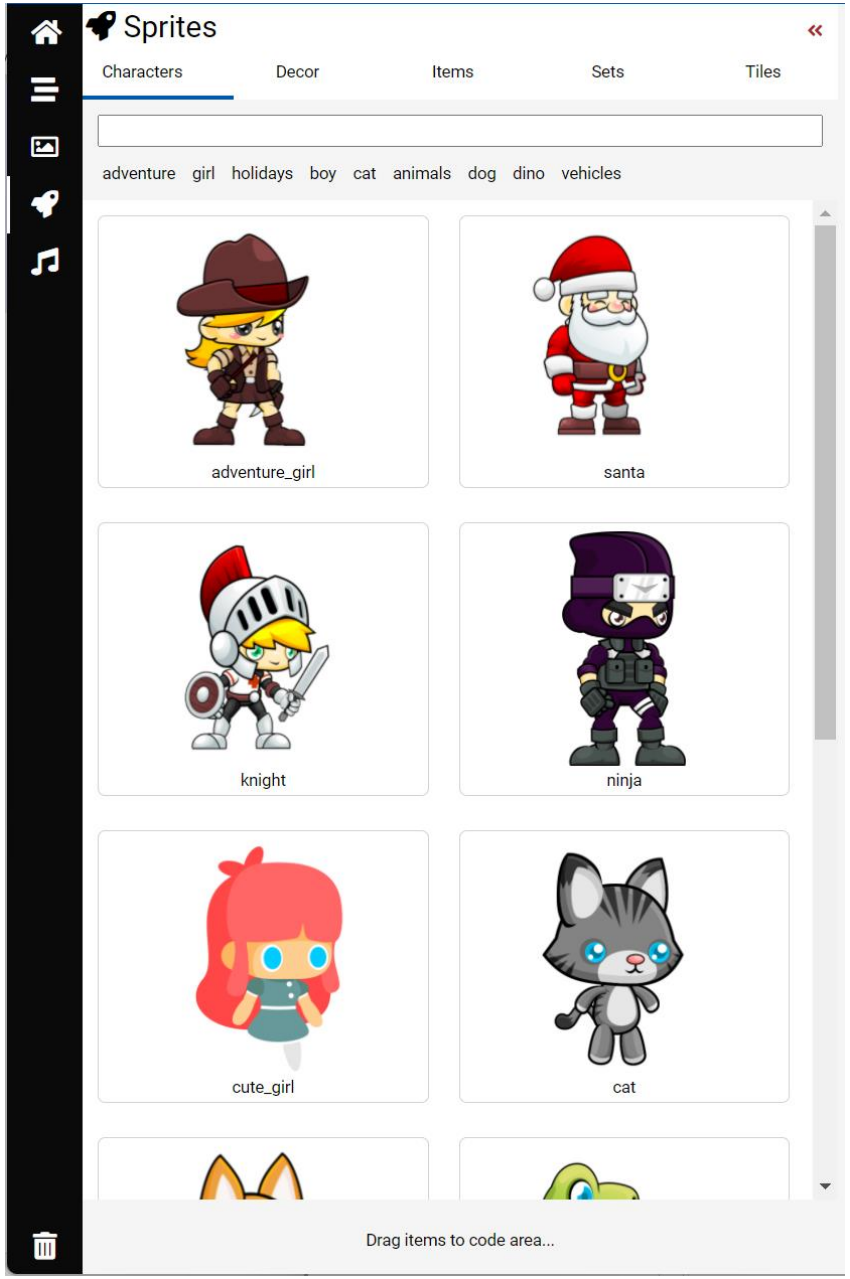
Here you'll see the output of your program.

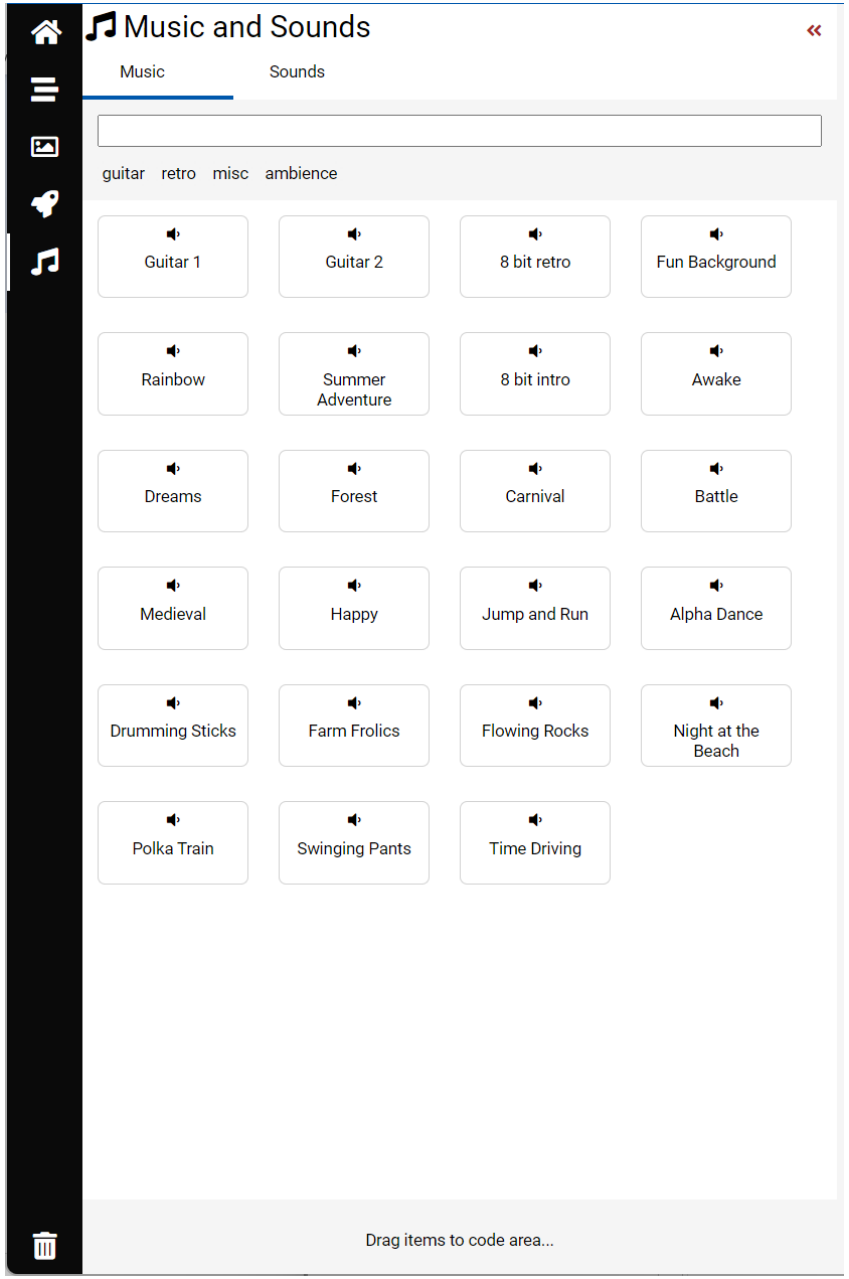
Code Editor

Type here your program

Exploring action bar







Let’s play with built-in assets

```
sprite('adventure_girl.idle', 400, 300, 0.5);
```

```
sprite('knight.idle', 400, 300, 0.5);
```

You don’t have to write the above code. Just drag and drop a sprite from the palette in the code editor to have this code generated for you.

Build a program using drag-and-drop

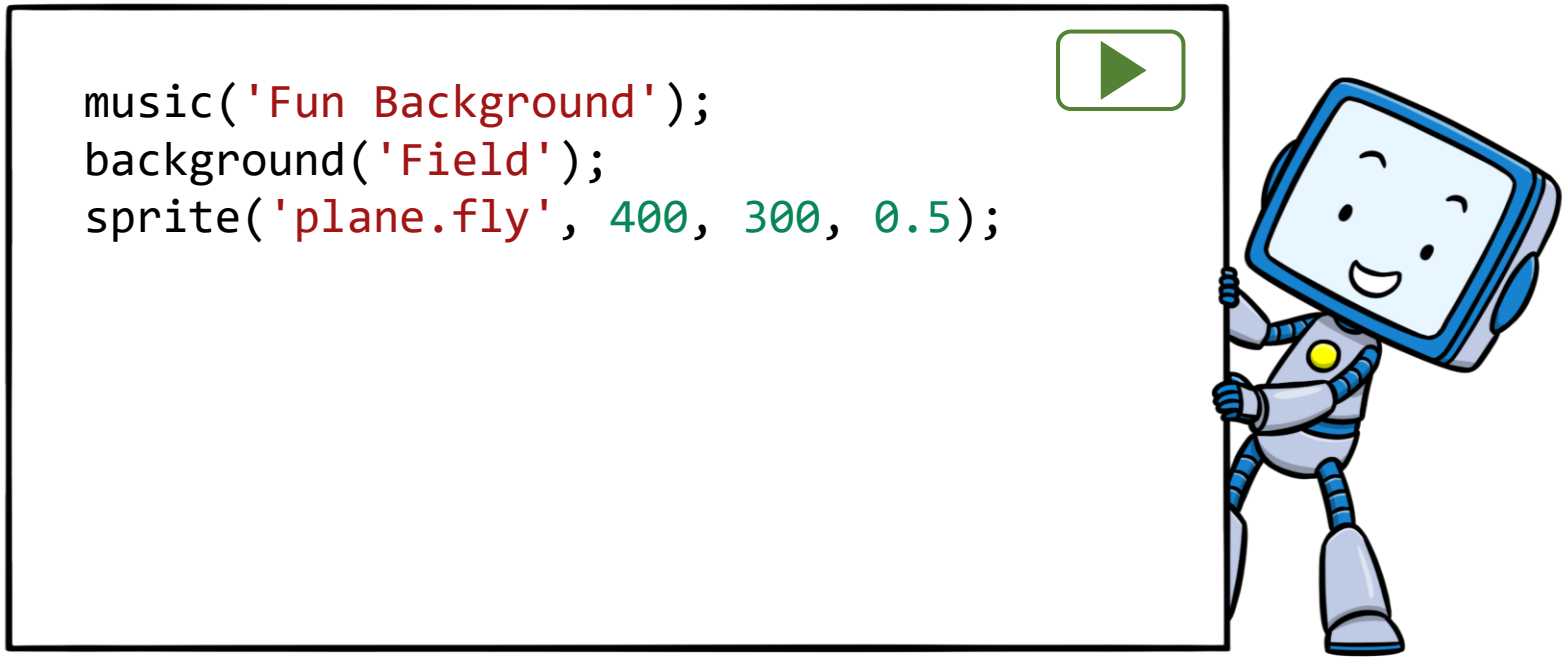
Step 1: Drag and drop a music file

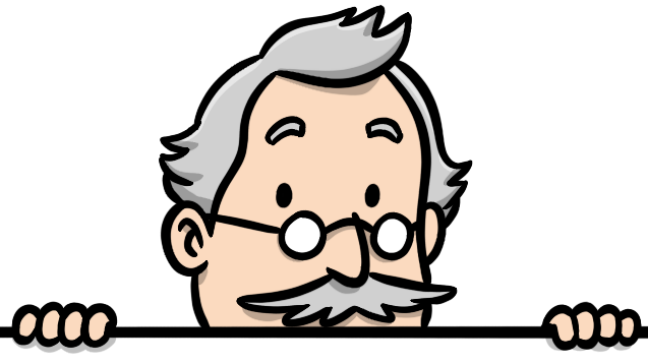
Step 2: Drag and drop a background image or color

Step 3: Drag and drop a sprite

Press “Play” when ready.

```
music('Fun Background');
background('Field');
sprite('plane.fly', 400, 300, 0.5);
```





```
sprite('adventure_girl.idle', 400, 300, 0.5);
```

Built-in instruction that asks computer to display a sprite on the canvas.

The actual sprite is specified in between quotes inside the parenthesis as a “parameter”.

Parameters of the instruction.

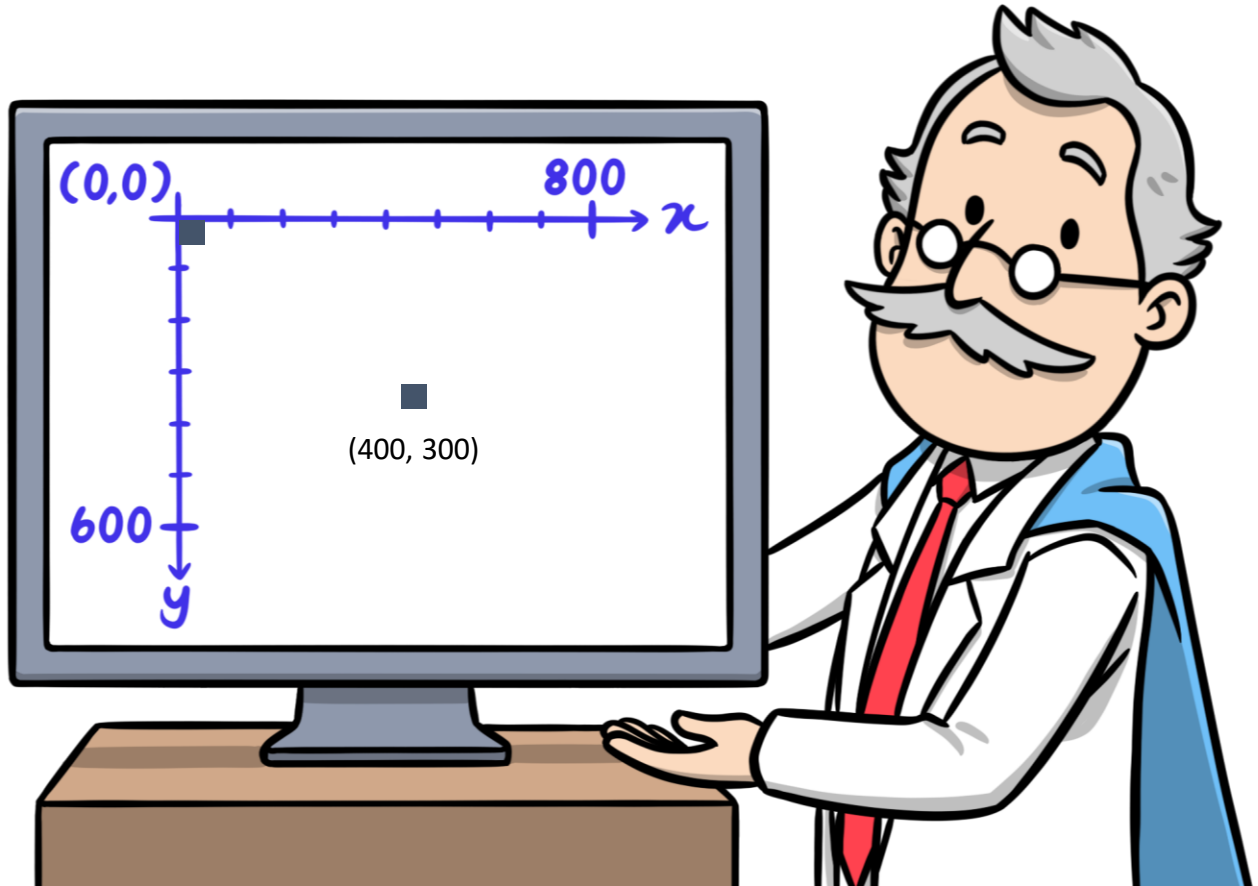
Parameters specify what sprite to display, where to display it on the canvas and at what coordinates.

Experiment: Try to change the 400 and 300 numbers with other numbers between 0 and 800.

Press “Play” / “Stop” after each modification to run / stop the program.

Output canvas

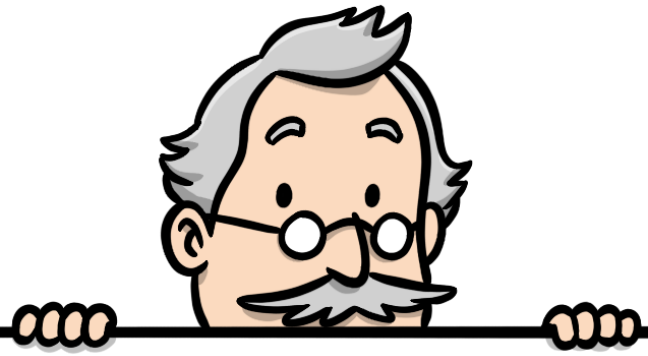
- Programs can write and draw on a graphical canvas of 800x600 pixels
- Origin is in the top-left corner
- Middle of the canvas is at about (400, 300)
- x coordinate goes from 0 to 800 (left to right)
- y coordinate goes from 0 to 600 (top to bottom)



Our first type-in program

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

- Type **carefully** this line in the code editor.
- Make sure you use the same **casing** as illustrated and include all the **punctuation** signs.
- When ready, press the **Play** / **Run** button



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

Built-in instruction that asks computer to draw a circle on the canvas.

codeguppy.com has many built-in instructions for different purposes (remember the sprite instruction used before).

Parameters of the instruction.

There are 3 parameters inside **parenthesis** and separated by **comma:**

- 400, 300 - coordinates of circle center
- 300 – radius of the circle

Try to modify the parameters of this instruction and notice the effect.

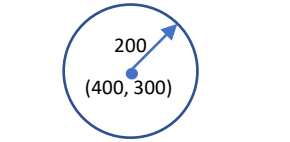
Don’t forget to press “Play” / “Stop” after each modification.

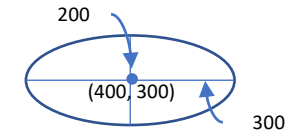
Multiple instructions in a program

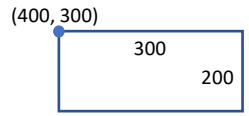
```
// Draw bear face
circle(400, 300, 200);

// Draw left ear
circle(250, 100, 50);
circle(270, 122, 20);

// Draw right ear
circle(550, 100, 50);
circle(530, 122, 20);

// Draw left eye
circle(300, 220, 30);
circle(315, 230, 10);

// Draw right eye
circle(500, 220, 30);
circle(485, 230, 10);

// Draw nose
circle(400, 400, 90);
circle(400, 350, 20);
```

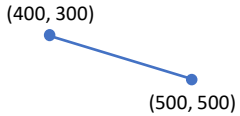
Other graphical instructions

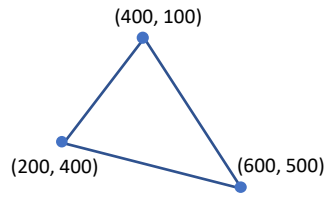
```
circle(400, 300, 200);

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);
```











Practicing graphical instructions

- **Exercise 1**: On a blank piece of paper draw an object / scene using only basic shapes such as circles, lines, rectangles. Convert that drawing into code!

OR

- **Exercise 2:** Download the “Draw with code” booklet from the downloads section. Choose any drawing, print the page and type-in the program.

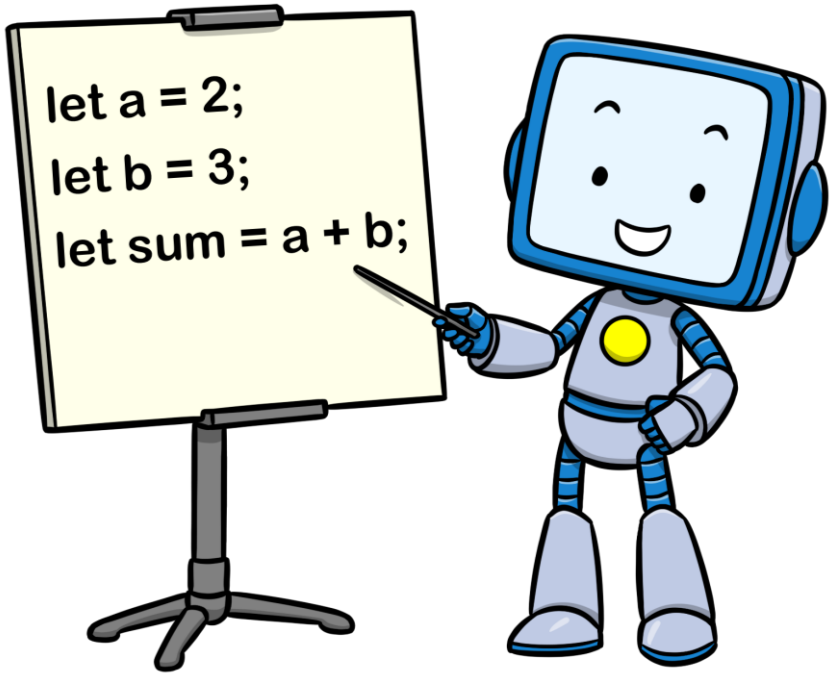
Note: Programs from the “Draw with code” booklet may have more instructions then presented here. Try to discover their purpose.

Introducing variables

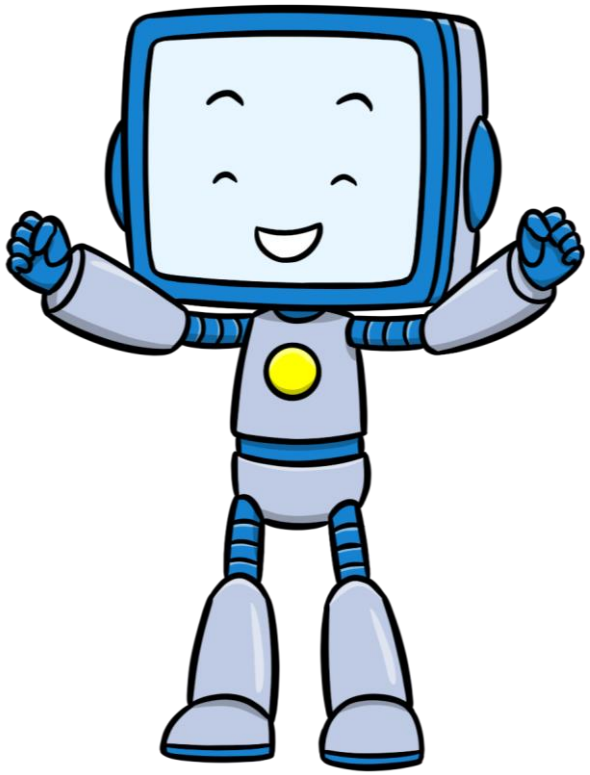
- JavaScript allows users to define variables to hold various values or the result of complex calculations
- You can name variables however you prefer… but don't start them with number or weird symbols
- Variables are super important in programming

```
let r = 100;
let r2 = 100 - 4;

circle(100, 100, r);
circle(100, 100, r2);
```

OR



Expressions and variables instead of scalars



You can let the computer do calculations. At the end of the day, your computer is also a powerful calculator.

```
circle(100, 100, 100);
circle(100, 100, 96);
```

```
circle(100, 100, 100);
circle(100, 100, 100 - 4);
```

```
let r2 = 100 - 4;
circle(100, 100, 100);
circle(100, 100, r2);
```

Building our own custom instructions

```
circle2();

function circle2()
{
    circle(100, 100, 100);
    circle(100, 100, 100 - 4);
}
```

1. We placed the two circle instructions inside a **code block** (delimited by curly brackets)

2. The block is the **body of function circle2**. Notice carefully the syntax.

3. The function circle2 is our new custom instructions. However, exactly like circle, it doesn't do anything until we **call** it.

Functions with parameters

```
circle2(400, 300, 100);

function circle2(x, y, r)
{
    circle(x, y, r);
    circle(x, y, r - 4);
}
```

Now this is a useful function! It can be used to draw a double-circle on the canvas.

We can control the position and radius of the double circle!

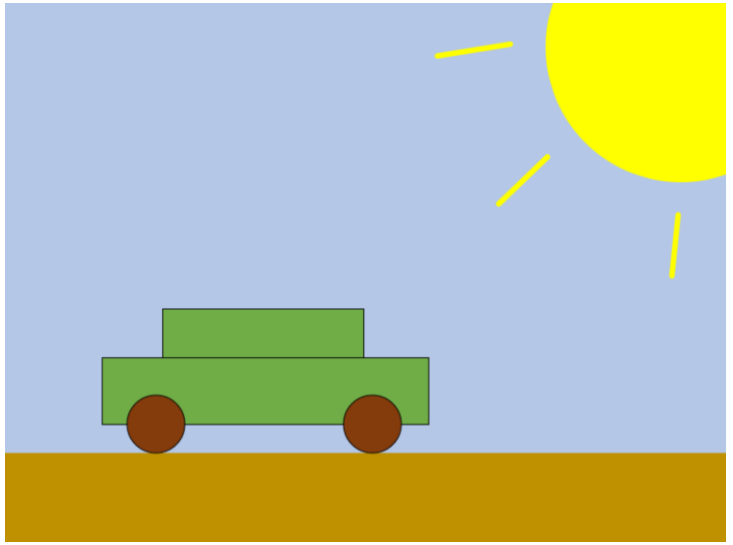
Exercise: Design your own custom instruction (e.g. function) to draw something useful.

Modularizing code with functions

- Functions can also be used to modularize big pieces of code into smaller pieces
- The smaller pieces can then be assembled or reused exactly like Lego pieces

Exercise: Copy or type-in the “Car” program from the “Draw with code” booklet / playground.

If you typed everything correctly you should see the following drawing when running the program.



Modularizing the code

```
background();
sun();
car();
```

This is main body of our new program. Of course, this program calls the specified functions.

In big programs, functions can be moved outside the main program in external module or libraries.

```
function background()
{
    // Background
    noStroke();
    fill(180, 199, 231);
    rect(0, 0, 800, 500);
    fill(191, 144, 0);
    rect(0, 500, 800, 100);
}
```

```
function sun()
{
    // Sun
    fill("yellow");
    circle(750, 50, 150);
    stroke("yellow");
    strokeWeight(6);
    line(480, 60, 561, 47);
    line(548, 224, 602, 172);
    line(740, 304, 747, 236);
}
```

```
function car()
{
    // Car
    stroke("black");
    strokeWeight(1);
    fill(112, 173, 71);
    rect(175, 340, 223, 54);
    rect(108, 394, 362, 74);
    fill(132,60,12);
    circle(168, 468, 32);
    circle(408, 468, 32);
}
```

Defining loop() function

```
function loop()
{
    background();
    sun();
    car();
}
```

```
function background()
{
...
}

function sun()
{
...
}

function car()
{
...
}
```

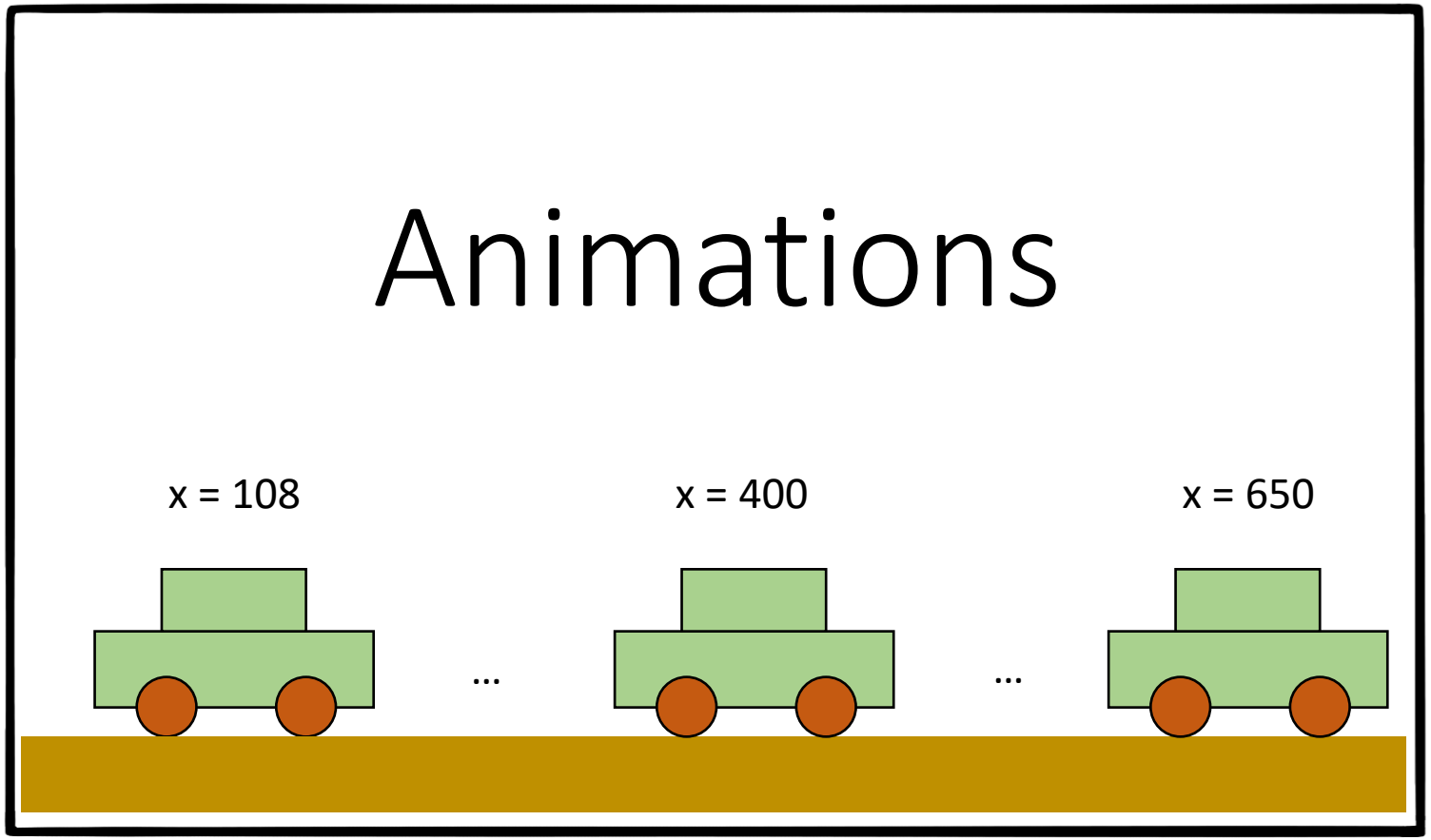
- loop() is a special function.
- We don’t have to call it.
- The system will call it automatically about 60 times / second.

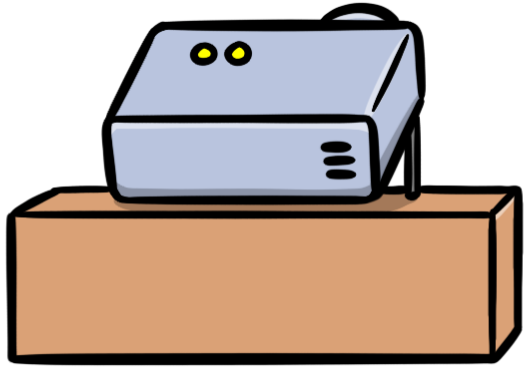
Parametrize car() function

```
car(108);

function car(x)
{
    // Car
    stroke("black");
    strokeWeight(1);
    fill(112, 173, 71);
    rect(x + 67, 340, 223, 54);
    rect(x, 394, 362, 74);
    fill(132,60,12);
    circle(x + 60, 468, 32);
    circle(x + 300, 468, 32);
}
```

We'll update function *car()* to accept a parameter *x* representing the *x* coordinate where the car will be drawn.

```
function loop()
{

Step 1: Clear the frame


Step 2: Redraw the
scene and car at
position x


Step 3: Increase
position x by 1

}
```





Increasing / decreasing variables

Increasing the value

```
x++;
```

OR

```
x += 1;
```

OR

```
x = x + 1;
```

Decreasing the value

```
x--;
```

OR

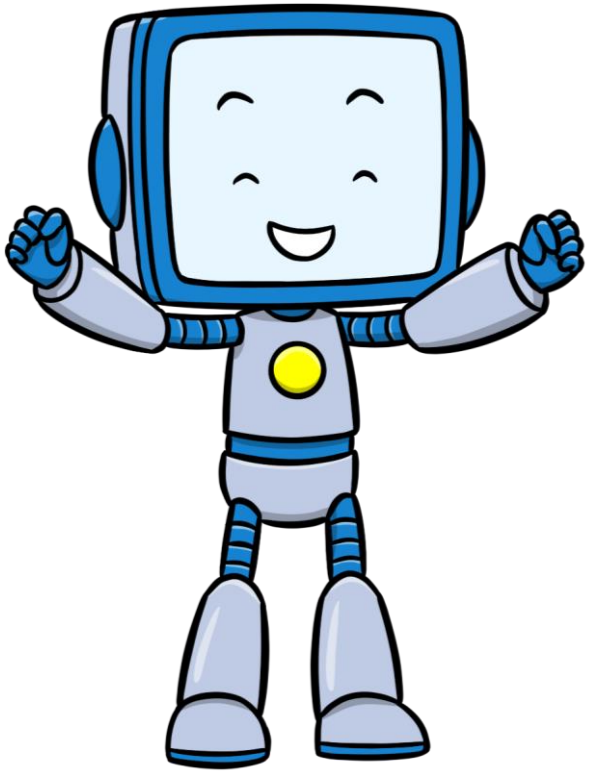
```
x -= 1;
```

OR

```
x = x - 1;
```

Note: x++ and x-- forms always modify the value by 1.
The other forms can be used to modify the value by an arbitrary amount.

Program listing



```
let x = 0;

function loop()
{
    clear();

    background();
    sun();
    car(x);

    x++;
}

function car(x)
{
    // Car
    stroke("black");
    strokeWeight(1);
    fill(112, 173, 71);
    rect(x + 67, 340, 223, 54);
    rect(x, 394, 362, 74);
    fill(132,60,12);
    circle(x + 60, 468, 32);
    circle(x + 300, 468, 32);
}
```

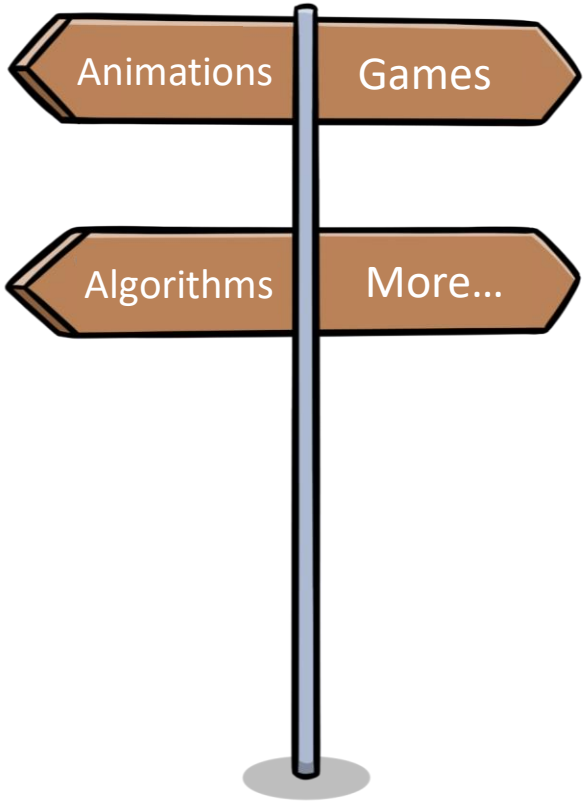
```
function background()
{
    // Background
    noStroke();
    fill(180, 199, 231);
    rect(0, 0, 800, 500);
    fill(191, 144, 0);
    rect(0, 500, 800, 100);
}

function sun()
{
    // Sun
    fill("yellow");
    circle(750, 50, 150);
    stroke("yellow");
    strokeWeight(6);
    line(480, 60, 561, 47);
    line(548, 224, 602, 172);
    line(740, 304, 747, 236);
}
```

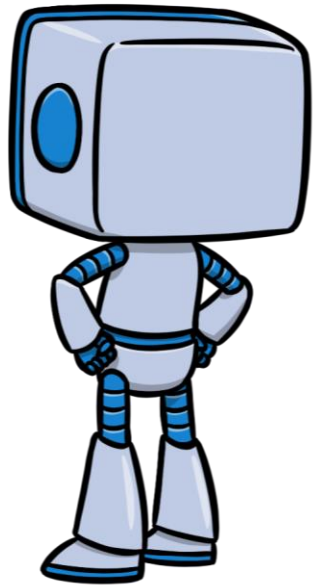
Next steps

- Complete the exercises presented today. Don’t be frustrated if you’ll encounter errors. Coding required **lots of practice** until you get comfortable writing programs.

- When working on a program, try to **run the program** from time to time to avoid accumulation of errors.

- **Explore the other tutorials and projects** from the codeguppy.com website and don’t forget to visit also the downloads section.





https://codeguppy.com
Free coding platform