Code.org notes and concepts

This guide is a reference for the programming concepts you've learned so far while using Code.org. While the programs you've been writing are relatively small, the concepts you're learning are at the foundation of software development.

Why should you learn the basics of programming? Learning to write code is similar to learning to read. In the same way that knowing how to read allows you to interact with the written world (books, recipes, social media posts, comics, text messages, etc), learning to write code allows you to interact with the world of computers and technology on your own terms.

- Avery N.

Definitions

Programs and code

A **program** is a list of instructions that a computer follows to accomplish some task. We call these instructions **code**. With Code.org, you'll focus on writing code that tells the computer how to display interactive animations and run games that you design.

Comments

Comments are notes in your code. Comments start with two forward slash symbols `//` and continue until the end of the line of code. The computer doesn't pay attention to comments when it runs your code; comments are just for explaining what code does.

```
// this is a comment
```

Sprites

A **sprite** is an image that is displayed when you run your program. A sprite can be a background, a character in a game, or any other image that you choose.

Functions and parameters

A **function** is an instruction that tells the computer to perform some task. For example, the `keyDown("key_name")` function tells the computer to check whether a certain key is pressed on the keyboard. When you use a function in your code, we say that you "call" the function.

A **parameter** is an extra piece of data that you give to a function to tell it how to perform a certain task. For example, when you use the `keyDown("key_name")` function to check if a certain key is pressed, you need to tell the function which key it should check by giving it the name of that key as the `"key_name"` parameter.

```
// check if the space key is pressed
```

```
keyDown("space");

// check if the left arrow key is pressed
keyDown("left");
```

Changing the `"key_name"` parameter that we give to the `keyDown("key_name")` function changes which key it checks.

Variables

Variables are names that you give to values in your program. You use the `var` keyword to make a new variable and assign it a value; then, you can use the variable name to refer to the value of the variable.

For example, we can assign numbers to variable names and then do math with those variables just like we would with the numbers.

```
var x = 1;
var y = 2;

// equals 3
x + y;
```

Conditionals

Conditional statements (also called **if** statements or **if else** statements) are used to restrict a piece of code so that it only runs when some condition is true.

```
// check if the space key is pressed
if (keyDown("space")) {
  // increase the size of the mario sprite if the space key is pressed
  mario.scale = mario.scale + 1;
}
```

In the example above, `keyDown("space")` is the condition that is being checked, and `mario.scale = mario.scale + 1` is the code that runs when that condition is true.

A conditional can have just an `if` component (code that runs when the condition is true), or it can have an `else` component as well (code that runs when the condition is false).

```
// check if the space key is pressed
if (keyDown("space")) {
  // increase the size of the mario sprite
  // if the space key is pressed
  mario.scale = mario.scale + 1;
} else {
```

```
  // decrease the size of the mario sprite
  // if the space key is not pressed
  mario.scale = mario.scale - 1;
}
```

Properties

A **property** is piece of data that is part of a larger piece of data. For example, if you have a sprite `mario` in your program, then the `x` and `y` position, `scale`, and `rotation` of the `mario` sprite are all **properties** of the sprite. You can access properties using the dot character, for example; `mario.x`, `mario.y`, `mario.scale`, and `mario.rotatation`.

Update a property:

```
// set the size of mario to 20% of the default size
mario.scale = 0.2;
```

Read a property:

```
// check if the x position of mario is greater than 200
if (mario.x > 200) {
  // do something when mario.x > 200
}
```

Methods

A **method** is a function that is attached to a piece of data, like a sprite. For example, if you have a sprite `mario` in your program, then the functions `mario.setAnimation("image_name")` and `mario.isTouching(other)` are methods of the `mario` sprite. You can access methods using the dot character.

```
// set the mario sprite to the image called "happy_mario"
mario.setAnimation("happy_mario");
```

`x` and `y` position

On the Code.org platform, the position of sprites and shapes is described in terms of `x` (the horizontal distance from the left edge of the board) and `y` (the vertical distance from the top of the board). The `x` and `y` positions each go from `0` to `400`. The middle of the board is `(x, y) = (200, 200)`.

Draw function

The **draw function** controls the dynamic behavior of your program: moving sprites, showing and hiding sprites, and responding to keyboard and mouse input. In general, you should put *setup code* (code that create sprites and sets their initial size and appearance) outside of the draw function and put *everything*

else inside the draw function.

How to

Set up a program from scratch

This is the basic code you should use when starting a project from scratch. Code to set up your program variables (like creating sprites with `createSprite(x, y)`) goes at the start of the program. Everything else should go in your draw function.

```
// set up your program and create your sprites here

function draw() {
  // clear the background
  background("white");

  // put any code for sprite movement, collision detection,
  // and responding to user input here

  // make the sprites appear
  drawSprites();
}
```

Create a sprite with `createSprite(x, y)`

Use the `createSprite(x, y)` function to make a new sprite. `createSprite(x, y)` takes has two parameters: the `x` and `y` values for the starting position of the sprite.

```
// create a sprite named "mario" in the middle of the screen
var mario = createSprite(200, 200);
```

Set the image for a sprite with `sprite.setAnimation("image_name")`

Use the `sprite.setAnimation("image_name")` method to set the image for a sprite. `sprite.setAnimation("image_name")` takes one parameter: the name of the image to use for the sprite. You will need to make sure that you've imported these images and given them names in the animation tab on Code.org in order to use them in your program.

```
var mario = createSprite(200, 200);

// set the mario sprite to the image called "angry_mario"
mario.setAnimation("angry_mario");

// note the difference between the name of the sprite (mario) and
// the name of the sprite image (angry_mario)
```

```
// to access the sprite later in the program use the
// variable name mario, not angry_mario
```

Create multiple sprites

You can create multiple sprites by using the `createSprite(x, y)` function multiple times. Make sure that you give each sprite a unique variable name.

```
// create a sprite named mario at position (x, y) = (100, 200)
var mario = createSprite(100, 200);
// set the image for mario to the image called "angry_mario"
mario.setAnimation("angry_mario");

// create a sprite named luigi at position (x, y) = (300, 200)
var luigi = createSprite(300, 200);
// set the image for luigi to the image called "happy_luigi"
luigi.setAnimation("happy_luigi");
```

Change the position of a sprite with `sprite.x` and `sprite.y`

You can change the position of a sprite by changing the `x` and `y` properties of the sprite.

```
// move the mario sprite to the bottom left corner of the board
mario.x = 0;
mario.y = 400;
```

Sprite movement with the counter pattern on `sprite.x` and `sprite.y`

You can make sprites move across the board using the **counter pattern**. The counter pattern works by setting the `x` or `y` position to itself plus or minus some small amount.

```
// move the mario sprite to the left by decreasing mario.x by 1
mario.x = mario.x - 1;

// move the mario sprite to the right by increasing mario.x by 1
mario.x = mario.x + 1;

// move the mario sprite up by decreasing mario.y by 1
mario.y = mario.y - 1;

// move the mario sprite down by increasing mario.y by 1
mario.y = mario.y + 1;
```

If you edit your program in text mode, you can also use these shortened versions of the counter pattern.

```
// move the mario sprite to the left by decreasing mario.x by 1
mario.x -= 1;

// move the mario sprite to the right by increasing mario.x by 1
mario.x += 1;

// move the mario sprite up by decreasing mario.y by 1
mario.y -= 1;

// move the mario sprite down by increasing mario.y by 1
mario.y += 1;
```

Sprite speed with the counter pattern on `sprite.x` and `sprite.y`

You can change the speed of the sprite movement created with the counter pattern by changing the amount that you add or subtract to the sprite position. Higher amounts produce faster movement.

```
// move the mario sprite to the left slowly
mario.x = mario.x - 1;

// move the mario sprite to the left quickly
mario.x = mario.x - 5;
```

Set the size of a sprite with `sprite.scale`

You can change the size of a sprite by changing the `scale` property of the sprite.

```
// set the sprite size to twice the default size
mario.scale = 2;

// set the sprite size to half the default size
mario.scale = 0.5;
```

Change the size of a sprite with the counter pattern on `sprite.scale`

You can change the size of a sprite gradually using the counter pattern on the `sprite.scale` property.

```
// gradually increase the size of the mario sprite
mario.scale = mario.scale + 0.1;

// gradually decrease the size of the mario sprite
```

```
mario.scale = mario.scale - 0.1;
```

Set the rotation of a sprite with `sprite.rotation`

You can change the rotation of a sprite by changing the `rotation` property of the sprite.

```
// set the sprite rotation to 90 degrees clockwise
mario.rotation = 90;

// set the sprite rotation to 90 degrees counter-clockwise
mario.rotation = -90;
```

Change the rotation of a sprite with the counter pattern on `sprite.rotation`

You can change the rotation of a sprite gradually using the counter pattern on the `sprite.rotation` property.

```
// gradually rotate the mario sprite clockwise
mario.rotation = mario.rotation + 1;

// gradually rotate the mario sprite counter-clockwise
mario.rotation = mario.rotation - 1;
```

Show and hide a sprite using `sprite.visibility`

You can make a sprite invisible by setting the `sprite.visible` property to `false`. You can make the sprite visible again by setting the `sprite.visible` property to `true`.

```
// hide the mario sprite
mario.visible = false;

// make the mario sprite visible again
mario.visible = true;
```

Stopping and looping a sprite around the edges of the board

If you make the sprite move in one direction, eventually it will move off the edge of the board and disappear.

```
// move mario to the left; mario will eventually
// disappear off the left side of the board
mario.x = mario.x - 5;
```

If you don't want this to happen, there are a couple of things you can do.

Option 1: stop the sprite

You can stop the sprite from moving off the board by checking its position and only running the counter pattern when the sprite hasn't reached the edge of the board.

```
// only move mario to the left if he hasn't gotten
// to the left edge of the board
// mario will stop moving when he gets to the left edge of the board
if (mario.x > 0) {
  // move mario to the left
  mario.x = mario.x - 5;
}
```

Option 2: loop the sprite

You can check if the sprite has reached the edge of the board and make the sprite loop around to the opposite edge of the board.

```
// move mario to the left
mario.x = mario.x - 5;

// if mario has reached the left edge of the board, move mario
// to the right edge of the board
// mario will keep moving left and looping around the board
if (mario.x < 0) {
  mario.x = 400;
}
```

Display text with `text("message", x, y)`

You can use the `text("message", x, y)` function to display a text on the board. The text can be a message in double quotes, or it can be a variable or a property.

```
// display the word hello at (x, y) = (100, 200);
text("hello", 100, 200);

// display the value of mario.x at (x, y) = (300, 400);
text(mario.x, 300, 400);
```

Check for key presses with `keyDown("key_name")`

You can use the `keyDown("key_name")` function to check if some key is pressed down. The `"key_name"` parameter controls which key the function checks.

```
// check if the space key is pressed
```

```
if (keyDown("space")) {
  text("space is pressed", 100, 200);
} else {
  text("space is not pressed", 100, 200);
}
```

To check if a letter key is pressed, use the name of the letter as the `"key_name"` parameter. For example, use `keyDown("a")` to check if the A key is pressed. Use `"space"` for the space bar, and `"left"`, `"right"`, `"up"`, `"down"` for the arrow keys.

Using `keyDown("key_name")` with the counter pattern

You can use the `keyDown("key_name")` function to with the counter pattern and conditionals to control the movement of sprites using the keyboard.

```
// check if the left arrow key is pressed
if (keyDown("left")) {
  // move the sprite named mario to the left
  mario.x = mario.x - 5;
}

// check if the right arrow key is pressed
if (keyDown("right")) {
  // move the sprite named mario to the right
  mario.x = mario.x + 5;
}

// check if the up arrow key is pressed
if (keyDown("up")) {
  // move the sprite named mario up
  mario.y = mario.y - 5;
}

// check if the down arrow key is pressed
if (keyDown("down")) {
  // move the sprite named mario down
  mario.y = mario.y + 5;
}
```

Check for mouse clicks with `mouseDown("leftButton")`

You can use the `mouseDown("leftButton")` function to check if the mouse is clicked. For simplicity, you should always use “leftButton” as the parameter to the function.

```
// check if the mouse is clicked
```

```
if (mouseDown("leftButton")) {
  text("mouse is clicked", 100, 200);
} else {
  text("mouse is not clicked", 100, 200);
}
```

Changing the appearance of a sprite using `sprite.setAnimation("image_name")`

You use `sprite.setAnimation("image_name")` to set the image for a sprite in the setup code before your draw function. You can also use `sprite.setAnimation("image_name")` in your draw function to change the image for a sprite after you create it.

```
// check if the mouse is clicked
if (mouseDown("leftButton")) {
  // change mario image to "happy_mario" when the mouse is clicked
  mario.setAnimation("happy_mario");
} else {
  // change mario image to "angry_mario" when the mouse is not clicked
  mario.setAnimation("angry_mario");
}
```

Count events with the counter pattern

In addition to movement, the counter pattern can also be used to count things (hence the name). To count things, you start by making a variable using `var` declaration to serve as the counter. Then, you can use the counter pattern to update the variable in response to some event.

```
// this goes before the draw function
// we are using a variable named clicks to keep track
// of the number of mouse clicks
var clicks = 0;

// this goes inside the draw function
if (mouseClick("leftButton")) {
  // update the click count each time there is a new mouse click
  clicks = clicks + 1;
  // display the number of mouse clicks on the screen
  text(clicks, 100, 200);
}
```

Common issues

Reusing the same variable names

```
// INCORRECT
// using the name player for two different sprites

var mario = createSprite(100, 200);
mario.setAnimation("happy_mario");

var mario = createSprite(300, 200);
mario.setAnimation("happy_luigi");

// CORRECT
// use a different name for each sprite

var mario = createSprite(100, 200);
mario.setAnimation("happy_mario");

var luigi = createSprite(300, 200);
luigi.setAnimation("happy_luigi");
```

Using the wrong variable name

```
var mario = createSprite(100, 200);

// INCORRECT
// the variable name has to match exactly

nario.setAnimation("happy_mario");

// CORRECT

mario.setAnimation("happy_mario");
```

***Note**: sometimes, if a variable name is wrong, Code.org will suggest that you need to wrap it in quotes. This suggestion is almost always wrong. You don't need to add quotes; you just need to make sure that the variable name you are using matches an actual variable in your program.*

Multiple draw functions

You should only have one draw function in your program.

```
// INCORRECT
// multiple draw functions

function draw() {
  if (keyDown("left")) {
```

```
    mario.x = mario.x - 5;
  }
}

function draw() {
  if (keyDown("right")) {
    mario.x = mario.x + 5;
  }
}

// CORRECT
// one draw function

function draw() {
  if (keyDown("left")) {
    mario.x = mario.x - 5;
  }

  if (keyDown("right")) {
    mario.x = mario.x + 5;
  }
}
```

Creating sprites in the draw function

You should always create sprites outside of the draw function.

```
// INCORRECT

function draw() {
  var mario = createSprite(200, 200);
}

// CORRECT

var mario = createSprite(200, 200);

function draw() {
  // ...
}
```

Forgetting to use `sprite.setAnimation("image_name")`

If you don't use `sprite.setAnimation("image_name")` to set the image for a sprite, then the sprite will show up as a gray square.

```
var mario = createSprite(200, 200);

// if you remove this line then the mario sprite will
// show up as a gray square
mario.setAnimation("happy_mario");
```

Missing quotes on parameters

When you use functions like `keyDown("key_name")`, `mouseClicked("leftButton")`, `sprite.setAnimation("image_name")` and `background("color_name")`, you usually want to pass a parameter surrounded by double quotes.

```
// INCORRECT
// missing quotes

keyDown(space);
background(white);
mouseClick(leftButton);
sprite.setAnimation(happy_mario);

// CORRECT

keyDown("space");
background("white");
mouseClick("leftButton");
sprite.setAnimation("happy_mario");
```

Syntax errors

If your code is written incorrectly, the computer won't be able to understand it. When this happens we call it a **syntax error**. Syntax errors are often caused by forgetting a symbol in your code, or adding an extra symbol in your code.

```
// INCORRECT
// missing the number after '+'

sprite.x = sprite.x + ;

// CORRECT

sprite.x = sprite.x + 1;

// INCORRECT
// missing the ')' after the function parameters
```

```
var mario = createSprite(200, 200;

// CORRECT

var mario = createSprite(200, 200);

// INCORRECT
// extra ',' between the function parameters

var mario = createSprite(200,, 200);

// CORRECT

var mario = createSprite(200, 200);
```

Forgetting the property name in the counter pattern

Make sure that you include the property name when you are trying to update some property of a sprite (like the `x`, `y`, `rotation`, `scale`, etc) when using the counter pattern. If you leave the property name out it will erase your sprite and result in confusing errors.

```
// INCORRECT
// missing the property name

mario = mario.x + 1;

// CORRECT

mario.x = mario.x + 1;
```

Redefining the wrong property name in the counter pattern

When you use the counter pattern, make sure that your are using the same property on both sides of the equals sign.

```
// INCORRECT
// using mario.x on one side and mario.y on the other side

mario.x = mario.y + 1;

// CORRECT

mario.x = mario.x + 1;
```

Using `var` in the wrong place

Sometimes when we set the value of a variable using `=` we use the `var` keyword, and sometimes we don't. How do we know when to use `var`?

- **Don't** use `var` when updating a property on a sprite (like `sprite.scale`)
- **Don't** use `var` when updating an existing variable (like `count = count + 1`).
- **Do** use var when making new variable, like with `createSprite(x, y)`.

```
// INCORRECT
// using `var` when updating a property on a sprite (like `sprite.scale`)

var mario.scale = 0.5;

// CORRECT

mario.scale = 0.5;

// INCORRECT
// using `var` to update an existing variable

var count = count + 1;

// CORRECT

count = count + 1;

// INCORRECT
// not using var when making new variable, like with `createSprite(x, y)`

mario = createSprite(200, 200);

// CORRECT

var mario = createSprite(200, 200);
```

Forgetting the `drawSprites()` function

If you don't include the `drawSprites()` function in your draw function, then the sprites you create will now show up. You normally want to put `drawSprites()` at the very end of your draw function.

```
// INCORRECT
// no drawSprites(), so sprites won't show up

function draw() {
```

```
  background("white");

  // ...
}

// CORRECT

function draw() {
  background("white");

  // ...

  drawSprites();
}
```

Forgetting the `background("color_name")` function

Normally, you need to call the `background("color_name")` function in your draw function in order to clear the board. You should call `background("color_name")` at the very beginning of your draw function. If you forget the `background("color_name")` then you will see overlapping copies of your sprites on the board.

```
// INCORRECT
// no background("color_name"), so sprites will overlap

function draw() {
  // ...

  drawSprites();
}

// CORRECT

function draw() {
  background("white");

  // ...

  drawSprites();
}
```

Reference

Functions

```
// play the sound that the url links to
playSound("url");

// check if the key with the name "key_name" is clicked or held down
keyDown("key_name");

// check if the key with the name "key_name" is clicked (but not held down)
keyWentDown("key_name");

// set the background color (use "white") to clear the board
background("color_name");

// return a random number between min and max
randomNumber(min, max);

// display a message on the board at the coordinate (x, y)
text("message", x, y);

// display the value of a variable on the board at the coordinate (x, y)
text(variableName, x, y);

// check if the left mouse button is clicked
mouseClick("leftButton");

// make the sprites appear (put this at the end of the draw function)
drawSprites();
```

Sprite properties

```
// the x position of the sprite named mario
mario.x;

// the y position of the sprite named mario
mario.y;

// the size of the sprite named mario
mario.scale;

// the rotation of the sprite named mario
mario.rotation;
```

Sprite methods

```
// change the image for the sprite named mario to the
```

```
// image called "happy_mario"
mario.setAnimation("happy_mario");

// check if the sprite named mario is touching the sprite named liugi
mario.isTouching(luigi);
```

World properties

```
// the x position of the mouse
World.mouseX;

// the y position of the mouse
World.mouseY;
```

Text editing

Remember that you can use keyboard shortcuts when editing your programs in text mode. You can use the undo command in block mode as well. These commands will save you lots of time when writing code; use them!

command	shortcut (chromebook)	shortcut (mac)
cut	ctrl + x	command + x
copy	ctrl + c	command + c
paste	ctrl + v	command + v
undo	ctrl + z	command + z

Example game code

In this game, the user controls a player character represented by an emoji. The emoji can be moved using the arrow keys, and will loop around the board if it goes off one of the edges. The game also includes a cake sprite placed at a random location on the board. When the emoji touches the cake, the emoji grows in size, and the cake jumps to a new location on the board. The game keeps track of the number of cakes eaten by the emoji and displays the count on the board. Once the player eats over 100 cakes, the count resets and the emoji shrinks back to the original size.

```
// player sprite
var player = createSprite(200, 200);
// set player image to "emoji_04_1"
player.setAnimation("emoji_04_1");
// set player size to 20% of default
player.scale = 0.2;

// cake sprite
var cake = createSprite(300, 300);
```

```
// set cake image to "shortcake_1"
cake.setAnimation("shortcake_1");
// set cake size to 20% of default
cake.scale = 0.2;

// number of cakes eaten
var count = 0;

function draw() {
  // clear the board
  background("white");

  if (keyDown("left")) {
    // move player to the left
    player.x = player.x - 10;
  }

  if (keyDown("right")) {
    // move player to the right
    player.x = player.x + 10;
  }

  if (keyDown("up")) {
    // move player up
    player.y = player.y - 10;
  }

  if (keyDown("down")) {
    // move player down
    player.y = player.y + 10;
  }

  // check if player is past the left edge of the board
  if (player.x < 0) {
    // loop player to the right edge of the board
    player.x = 400;
  }

  // check if player is past the right edge of the board
  if (player.x > 400) {
    // loop player to the left edge of the board
    player.x = 0;
  }

  // check if player is past the top edge of the board
```

```
  if (player.y < 0) {
    // loop player to the bottom edge of the board
    player.y = 400;
  }

  // check if player is past the bottom edge of the board
  if (player.y > 400) {
    // loop player to the top edge of the board
    player.y = 0;
  }

  // check if player sprite and cake sprite are touching
  if (player.isTouching(cake)) {
    // increase the size of the player sprite
    player.scale = player.scale + 0.05;
    // set the cake x and y position to a random point on the board
    cake.x = randomNumber(0, 400);
    cake.y = randomNumber(0, 400);
    // increase the count by 1
    count = count + 1;
  }

  // if the count gets over 100
  if (count > 100) {
    // reset the count to zero
    count = 0;
    // reset the player size
    player.scale = 0.2;
  }

  // print the count in the bottom left corner of the board
  text(count, 20, 380);
  // make the sprites appear
  drawSprites();
}
```