# GD 3
# Costumes For You

**The Falling Dot in GD 2 was simply a circle**. It would be nice if we could change the look of the falling dot so that a **costume is used as a skin**. Programming languages have an array as a group of storage elements. **We could use arrays to store the different costumes and then choose a single costume for use by using a variable as a digital key.**

**Game Falling Dots With Costumes** – a dot can change position and color. In this lesson, **we will focus on changing the costume of the falling dot** and we will use it array to implement costume change.

1. Array
2. Array manipulation
3. Digital key
4. `.push()` into array

---

## 0. ORGANIZATION OF GAME OBJECTS

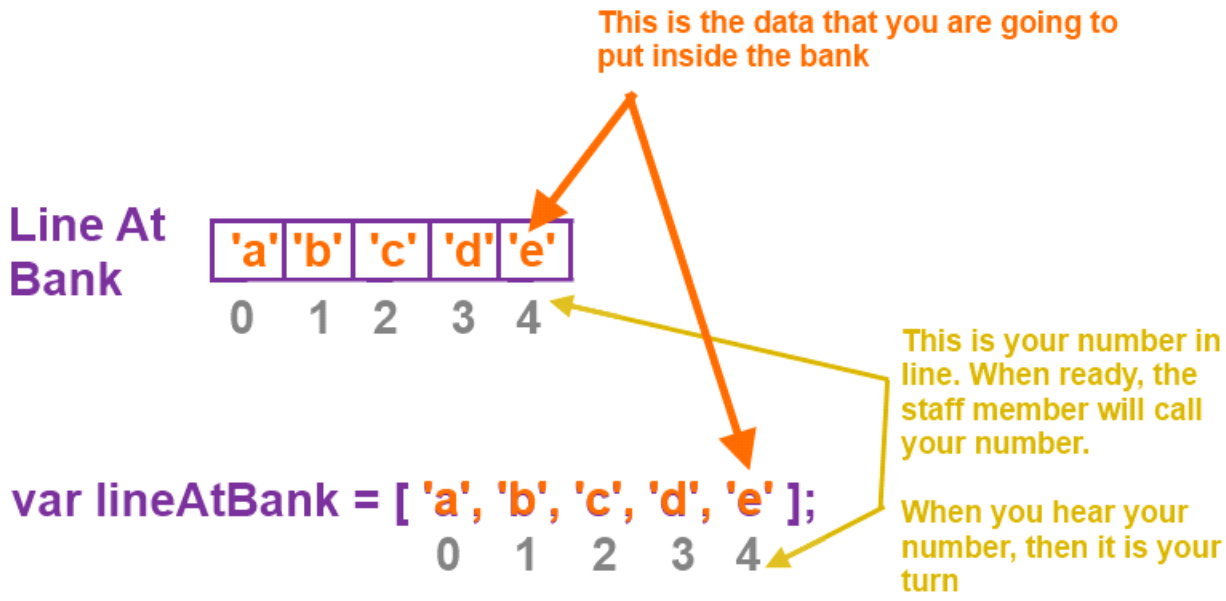
### 0.1. Situation

Currently, the falling dot can change colors but doesn't have a costume. **Let's customize our falling dot to have a costume and we get to change the costume when the player successfully clicks on the hit box of the dot.**

**Remember in Scratch, our game sprite can have multiple costumes.** How can we implement a costume feature so that we can pick which costume to use by looking at the costume number? The answer is to use an array and the lesson on array is below.

**Another way to think about it is that an array is basically a line.**

When a staff member is done helping one person in line, the staff member will move onto the next position and help the person in that position. So, if the staff member is done helping the person in position 0, the staff member will then move to the next position in the line ( the next position after 0 is position 1 ) and help the person in position 1. **The position that the staff member is on is called the index.**




Notice that an array is a group of neighboring memory elements and the word "**neighboring**" means that each memory element within the array is in sequential order ( 0, 1, 2, 3, and etc ). The order **ALWAYS** starts at **number 0**.

With an array, we access the array by using a special format.

arrayName**[**index**]**;

The **total length** of the array can be found by using arrayName.length.

The **first index** is **always 0** and the **last index** is arrayName.length - 1

lineAtBank**[**0**]**

lineAtBank.length - 1

lineAtBank.length

### 0.2. access group using array name

Since all memory elements are in the same group, we can move all memory elements of the group by simply using the name of the array.

var **lineAtBank** = [ 'a', 'b', 'c', 'd', 'e' ];

var **myArr** = **lineAtBank**;

The code above transfers the **ENTIRE CONTENT** of the array named **lineAtBank** into the array named **myArr**. The key operation is the assignment operator, which is used to transfer data from the right side of the assignment operator to the left side of the assignment operator.

### 0.3. access single element using square bracket

What if we want to access **ONLY A SINGLE MEMORY ELEMENT** within the array? **The answer is to use square brackets, which look like this [ ]**

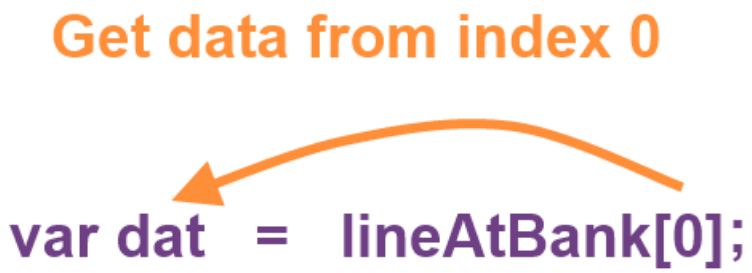
lineAtBank**[**1**]** = 'v';

The **square brackets** are used to select **ONLY A SINGLE MEMORY ELEMENT** within the array. Inside the square brackets is the index ( ie. inside the square brackets is the position in the line we are currently on).

a. The code above uses square brackets to select **index 1** of the array ➔ the code above means we are at **position 1** and putting the data 'v' into **position 1.**

After the line of code above is finished executing, the content of the array named dataGroup is updated to be `var lineAtBank = [ 'a', 'v', 'c', 'd', 'e' ];`

### 0.4. get data from a single array index



**0.5. put data into a single array index**


**0.6. digital key**

All of the examples from above put a number inside the square bracket. This is called a **hardcoded index**. We can't change the index because we are not using a variable.

```
lineAtBank[0]
```

Can we replace the number with a variable and make a generic index? The answer is yes and using variable as a generic index makes the variable a digital key.

```
var pos = 0;

lineAtBank[pos] = 'v';

pos +=1;

lineAtBank[pos] = 'z';
```

**Explanation**

The variable pos has the value of 0 stored inside of it. The code above puts the data 'v' into the memory element at the index found in variable **`pos`**. So, we are putting the data 'v' into the memory element at index 0. Since we are using the variable **`pos`** as an index, then **`pos`** becomes a digital key.

Next, we use the special ability of a variable to retain information and use this ability to perform index manipulation. Variables store data and retains data until new data is loaded into it. We execute the line of code **`pos += 1`**, which add 1 to the current value of **`pos`**. The variable **`pos`** has the value of 0 inside of it and we add 1 to it.

So now, 0 + 1 = 1. We then store the new value of 1 into **`pos`** using the assignment operator. The variable **`pos`** now has the value of 1 stored inside of it. We then use **`pos`** at an index and put the data 'z' into memory element at
index 1.

### 0.7. reading a "for" loop using an array

```
    var sum    = 0;
    var numArr = [ 1, 2, 10, 11, 50, 100 ];
// -- position     0  1  2   3   4   5
```

Position 0 has the data 1 store inside of it

begin
a. begin only happens once

end

jump by
a. if not at the end, then jump by this much

```
for( var i = 0; i < numArr.length; i++ )
{
    sum += numArr[i];
}
// -- next
return sum/numArr.length;
```

loop body

i++ --> jump by 1 position
i+= 2 --> jump by 2 position

**Reading a "for" loop**

1. set i = 0
2. check if at the end
   Yes, go to line next and continue
   No, run line of code in loop body
3. jumpy by i++

## Array Review

**var myTabs = [ "Home", "Projects", "Resume", "College" ];**

1. what is the first index number?
2. what is the data stored inside index 0?
3. what is the length of the array? Give a math equation instead of a number.

4. how do I select **ONLY ONE** array element?

## 1. COSTUME ARRAY

**Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* =====================================
* varibles for costume array and index
*
*
* ===================================== */
var
dotImgArr = [],     // -- empty array
costumeIndex;
```

**Explanation**

In the code above, we are declaring an array that is named **`dotImgArr`** and making it an empty array. The square bracket on the right side with nothing in between means empty array. We will fill in this empty array with costumes later on.

Next, we declare another variable named **`costumeIndex`** and it will be used as a digital key. **Remember, a digital key is a marker of our position within the array.**

## 2. PUSH INTO ARRAY

**Write the code below in between <script> </script>.** The **large, green banner** is your landmark. **Go to the coding website and look for it.** Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* =====================================
* Push costume into array
*
*
=========================================*/
costumeIndex = 0;

var imgObj = new Image();
imgObj.src = "https://vuongducnguyen.com/images/asteroid.png";
dotImgArr.push( imgObj );

imgObj = new Image();
imgObj.src = "https://vuongducnguyen.com/images/packM.png";
dotImgArr.push( imgObj );

imgObj = new Image();
imgObj.src = "https://vuongducnguyen.com/images/green_line.png";
dotImgArr.push( imgObj );
```

**Explanation**

From Step 1, we created an empty array. How do we put data into an array? There are many ways to accomplish this. For the code above, the `.push()` method is used to put data into an array. **The `.push()` method is a special function because it puts data into the array in sequential order and the data is put into the array at the end.** This is equivalent to getting in line at the end ( getting in line at the front is cutting → BAD!!! )

What does this mean? **It means that data is put into the array at the next available position.** So, if our current position is 0 and we then perform a `.push()`, then the data is put into the next available position. **Since position 0 is taken, this means that the next available position is 1. So, the data is put into the array at position 1.**

## 3. NEW COSTUME

**Write the code below in between <script> </script>.** The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* ======================================
* Player catch dot and gets new costume
*
*======================================*/
costumeIndex++;
if( costumeIndex >= dotImgArr.length )
{
      costumeIndex = 0;
}
```

**Explanation**

If a player moves the mouse cursor over to the falling dot and then clicks on it while it is falling, the mouse's (x,y) position coordinate is within the hit box of the falling dot. This is a successful hit and the falling dot will change costumes. The code in **blue** changes the position marker to the next costume.

For example, if `costumeIndex` has the value of 1, then **after the code in blue executes**, then `costumeIndex` will have a value of 2. This means that we are selecting costume at position 2 for use.

**An array is important because it holds the available costume for use.** We don't have an infinite number of costumes. Instead, we only use the costumes that are already stored within the array named `dotImgArr`.

If we use every costume within the array, then we go back to costume 0. The code in **color orange** checks if we have used every costume in the array. If so, then choose costume 0 by loading the value of 0 into the position marker ( or digital key ) named `costumeIndex`.

## 4. USE COSTUME USING DIGITAL KEY "costumeIndex"

**Write the code below in between <script> </script>.** The **large, green banner** is your landmark. Go to the coding website and look for it. Next, write the code below underneath the **large, green banner**. **Write all of it, color code is for explanation.**

```
/* =========================
* falling rain
*
*
*========================== */
cYPos -= cYStep * cYDir;

ctx.fillStyle = cColor;
ctx.beginPath();
ctx.drawImage( dotImgArr[costumeIndex], cXPos, cYPos, cRad*2, cRad*2 );
ctx.fill();
```

**Explanation**

The code in **green** draws the falling dot as an image. From **Step 2**, we put 3 images into the array and so which image is currently being used? Remember that an array is a group of storage elements. **How do we pick ONE ELEMENT of the array**? **We use square brackets.** The number between the square brackets is the costume number that is chosen.

We are using a position marker ( or digital key ) of **costumeIndex** to select a costume.

Remember that if we don't use a position marker and write `dotImgArr[3]`, **then we are stuck at costume 3 forever.**

**However, if we use a position marker, then we can put new data into the variable and that allows us to use the variable as a position marker to choose different costume numbers.**

**JS Challenge**

1. **Go back to Step 2** and use **.push()** to add more costumes into the array. Use the image link https://vuongducnguyen.com/images/ to select a costume and use **.push()** to put the costume into the array.

2. Add 3 more costumes into the array

3. Test the game and see if the costumes appear