[Frequently Asked Questions | Codecademy](https://www.codecademy.com/forum_questions/533857ac631fe9794700a672)

**How does alphabet.js work?**

In alphabet.js there is defined only one object called document.alphabet.

document.alphabet is a collection of objects (representing single characters) with names coded in hexadecimal values (in the pattern A + hexValue). The code in this script is organized in this manner:

document.alphabet = {

A79: {

W: 75,

P: [[64, 89, 9, -102], [57, 103, 9, -102], [5, 89, 9, -79], [16, 104, 8, -35], [51, 122, 8, -35], [23, 118, 8, -35], [31, 133, 8, 50], [46, 136, 8, 50], [34, 153, 8, 69], [28, 168, 7, 112], [21, 183, 7, 112]]

},

A78: {

W: 85,

P: [[10, 148, 8, -103], [21, 137, 8, -92], [33, 125, 7, -79], [50, 124, 7, -35], [58, 135, 7, -35], [68, 148, 7, -35], [40, 111, 7, 51], [33, 103, 7, 51], [21, 86, 7, 51], [56, 106, 7, 51], [67, 92, 7, 112]]

},

A77: {

W: 98,

P: [[24, 148, 10, -102], [67, 151, 10, -102], [34, 133, 9, -36], [61, 134, 9, -36], [76, 132, 8, -36], [16, 130, 8, -36], [14, 112, 7, 50], [40, 118, 7, 50], [58, 117, 7, 50], [82, 117, 7, 83], [86, 100, 7, 83], [51, 101, 7, 83], [11, 95, 7, 83]]

},

// more more more

A3d: {

W: 73,

P: [[11, 91, 9, -50], [27, 90, 9, -50], [44, 90, 9, -50], [59, 90, 9, -50], [11, 127, 6, 69], [21, 128, 6, 69], [33, 129, 6, 69], [44, 129, 6, 69], [53, 129, 6, 69]]

}

};

[Here you can find list with hexadecimal value of every printable character](http://en.wikipedia.org/wiki/ASCII#ASCII_printable_characters)

* A79 - character with hex code 79 represents a y
* A78 - character with hex code 78 represents an x
* A77 - character with hex code 77 represents a w
* A3d - character with hex code 3d represents an =

Every letter has two properties, W and P.

W stands for “width”. A higher value of W means the character is broader. That is why W of A21 (!) is so small (35) and A51 (Q) is so large (125).

P is a collection of points (single dots on your animation). Every point has 4 values, for example [64, 89, 9, -102]. First two values are coordinates (x = 64, y = 89), they have an impact on where the dot is located. Third value (9) specifies size of the dot. Last value (-102) is not used in our project, but originally it was defining the tone selected by user color (hue).

|  |
| --- |
| // Vector is a data structure used to represent a point in 3d space |
|  | function Vector(x, y, z) { |
|  | // properties x, y, z represent each coordinate of the point |
|  | this.x = x; |
|  | this.y = y; |
|  | this.z = z; |
|  | // the set method is used to change the x and y coordinates of the given point |
|  | this.set = function (x, y) { |
|  | this.x = x; |
|  | this.y = y; |
|  | }; |
|  | } |
|  |  |
|  | // PointCollection is a data structure used to represent all points forming our animation |
|  | function PointCollection() { |
|  | /\* the mousePos property stores the coordinates of the cursor, |
|  | \* its initial value is a point in the upper left corner of the 2d plane |
|  | \*/ |
|  | this.mousePos = new Vector(0, 0); |
|  |  |
|  | /\* the pointCollectionX and pointCollectionY properties store |
|  | \* an additional, random deviation to the position of the point, |
|  | \* their initial value is 0 |
|  | \*/ |
|  | this.pointCollectionX = 0; |
|  | this.pointCollectionY = 0; |
|  |  |
|  | /\* the points property stores all of the points forming our animation, |
|  | \* its initial value is an empty array |
|  | \*/ |
|  | this.points = []; |
|  |  |
|  | // the update method is used to track the position of the cursor and accordingly influence each point |
|  | this.update = function () { |
|  | // for every element in the points array (...) |
|  | for (var i = 0; i < this.points.length; i++) { |
|  | /\* Assign: |
|  | \* - to variable point: the current point (the element at index i in the points array) |
|  | \* - to variable dx: the horizontal distance between the cursor and the current point |
|  | \* - to variable dy: the vertical distance between the cursor and the current point |
|  | \* - to variable d: the distance in a straight line between the cursor and the current point, |
|  | \* this variable is calculated using the Pythagorean theorem |
|  | \*/ |
|  | var point = this.points[i]; |
|  | var dx = this.mousePos.x - point.curPos.x; |
|  | var dy = this.mousePos.y - point.curPos.y; |
|  | var d = Math.sqrt((dx \* dx) + (dy \* dy)); |
|  |  |
|  | /\* Statements below are assignation operations combined with ternary operators [add\_doc] |
|  | \* If the distance between the cursor and the current point is less than 150 then assign |
|  | \* to the property targetPos [add\_lines] of the current point the difference between the |
|  | \* current position of the point and the distance between the cursor and the current position |
|  | \* of the point. [add\_better\_explanation] [optimize\_code] |
|  | \* Otherwise assign to the property targetPos [add\_lines] of the current point the original |
|  | \* position of this point. |
|  | \*/ |
|  | point.targetPos.x = d < 150 ? point.curPos.x - dx : point.originalPos.x; |
|  | point.targetPos.y = d < 150 ? point.curPos.y - dy : point.originalPos.y; |
|  |  |
|  | // trigger the update method [add\_line\_numbers] for the current point |
|  | point.update(); |
|  | } |
|  | }; |
|  |  |
|  | /\* the shake method is used to shake our collection of points. This method is a significant part of the |
|  | \* bounceName function [add\_lines] we used, for example, in this exercise: |
|  | \* http://www.codecademy.com/courses/animate-your-name/2/7 |
|  | \*/ |
|  | this.shake = function () { |
|  | // for every element in the points array (...) |
|  | for (var i = 0; i < this.points.length; i++) { |
|  | /\* Assign: |
|  | \* - to variable point: the current point (the element at index i in array points) |
|  | \* - to variable dx: the horizontal distance between the cursor and the current point |
|  | \* - to variable dy: the vertical distance between the cursor and the current point |
|  | \* - to variable d: the distance in a straight line between the cursor and the current point, |
|  | \* this variable is calculated using the Pythagorean theorem |
|  | \*/ |
|  | var point = this.points[i]; |
|  | var dx = this.mousePos.x - point.curPos.x; |
|  | var dy = this.mousePos.y - point.curPos.y; |
|  | var d = Math.sqrt((dx \* dx) + (dy \* dy)); |
|  |  |
|  | // if the distance between the cursor and the current point is less than 50 (...) |
|  | if (d < 50) { |
|  | /\* (...) Assign to the pointCollectionX and pointCollectionY properties two random integer numbers |
|  | \* from set [-2, -1, 0, 1, 2] |
|  | \*/ |
|  | this.pointCollectionX = Math.floor(Math.random() \* 5) - 2; |
|  | this.pointCollectionY = Math.floor(Math.random() \* 5) - 2; |
|  | } |
|  |  |
|  | /\* trigger the draw method [add\_lines] for the current point with the parameters |
|  | \* pointCollectionX and pointCollectionY which affect the position of the point |
|  | \*/ |
|  | point.draw(bubbleShape, this.pointCollectionX, this.pointCollectionY); |
|  | } |
|  | }; |
|  |  |
|  | // the draw method is used to draw our collection of points |
|  | this.draw = function (bubbleShape, reset) { |
|  | // for every element in the points array (...) |
|  | for (var i = 0; i < this.points.length; i++) { |
|  | // (...) assign to the variable point the current point (the element at index i in the points array) |
|  | var point = this.points[i]; |
|  |  |
|  | // if the current point does not exist (...) |
|  | if (point === null) |
|  | // (...) go to the next iteration, next point |
|  | continue; |
|  |  |
|  | // if the reset property of the window object is set to true (...) |
|  | if (window.reset) { |
|  | // (...) assign default, initial values to the variables as listed below |
|  | this.pointCollectionX = 0; |
|  | this.pointCollectionY = 0; |
|  | this.mousePos = new Vector(0, 0); |
|  | } |
|  |  |
|  | // trigger the draw method for the current points [add] |
|  | point.draw(bubbleShape, this.pointCollectionX, this.pointCollectionY, reset); |
|  | } |
|  | }; |
|  | } |
|  |  |
|  | // Point is a data structure used to represent single points / bubbles in our animation |
|  | function Point(x, y, z, size, color) { |
|  | /\* the curPos property stores the current position of our bubble in 3d space, |
|  | \* its initial value is equal to the coordinates defined in alphabet.js (parameters x, y, z) |
|  | \*/ |
|  | this.curPos = new Vector(x, y, z); |
|  | // the color property stores the color of our bubble defined by us in main.js |
|  | this.color = color; |
|  |  |
|  | // load these settings from document [add\_line\_numbers] |
|  | this.friction = document.Friction; |
|  | this.rotationForce = document.rotationForce; |
|  | this.springStrength = document.springStrength; |
|  |  |
|  | // the originalPos property stores a Vector (point) with the coordinates defined in alphabet.js (parameters x, y, z) |
|  | this.originalPos = new Vector(x, y, z); |
|  | // the basic values of radius and size is a value defined in alphabet.js (parameter size) |
|  | this.radius = size; |
|  | this.size = size; |
|  | // the targetPos property stores the direction where bubble goes, initial value is equal to originalPos |
|  | this.targetPos = new Vector(x, y, z); |
|  | // velocity in our script is represented by a vector, initial velocity is equal to 0 |
|  | this.velocity = new Vector(0.0, 0.0, 0.0); |
|  |  |
|  | /\* the update method is used to update the properties of the Point object |
|  | \* in accordance with the current situation |
|  | \*/ |
|  | this.update = function () { |
|  | var dx = this.targetPos.x - this.curPos.x; |
|  | var dy = this.targetPos.y - this.curPos.y; |
|  | var ax = dx \* this.springStrength - this.rotationForce \* dy; |
|  | var ay = dy \* this.springStrength + this.rotationForce \* dx; |
|  |  |
|  | this.velocity.x += ax; |
|  | this.velocity.x \*= this.friction; |
|  | this.curPos.x += this.velocity.x; |
|  |  |
|  | this.velocity.y += ay; |
|  | this.velocity.y \*= this.friction; |
|  | this.curPos.y += this.velocity.y; |
|  |  |
|  | /\* Assign: |
|  | \* - to variable dox: the horizontal distance between the original and the current position of point |
|  | \* - to variable doy: the vertical distance between the original and the current position of point |
|  | \* - to variable d: the distance in a straight line between the original and the current |
|  | \* position of the point, this variable is calculated using the Pythagorean theorem |
|  | \*/ |
|  | var dox = this.originalPos.x - this.curPos.x; |
|  | var doy = this.originalPos.y - this.curPos.y; |
|  | var d = Math.sqrt((dox \* dox) + (doy \* doy)); |
|  |  |
|  | this.targetPos.z = d / 100 + 1; |
|  | var dz = this.targetPos.z - this.curPos.z; |
|  | var az = dz \* this.springStrength; |
|  | this.velocity.z += az; |
|  | this.velocity.z \*= this.friction; |
|  | this.curPos.z += this.velocity.z; |
|  |  |
|  | this.radius = this.size \* this.curPos.z; |
|  | if (this.radius < 1) this.radius = 1; |
|  | }; |
|  |  |
|  | // the draw method is used to draw a single point / bubble |
|  | this.draw = function (bubbleShape, dx, dy) { |
|  | // set the fill color to the color of the bubble |
|  | ctx.fillStyle = this.color; |
|  |  |
|  | if (bubbleShape == "square") { |
|  | // begin path |
|  | ctx.beginPath(); |
|  | /\* To draw a rectangle filled with the current fillStyle we use |
|  | \* fillRect(x, y, width, height) |
|  | \* where x, y is the upper, left corner of the rectangle. |
|  | \* In this case, we use this.radius \* 1.5 for width and height, and |
|  | \* the upper, left vertex has coordinates (this.curPos.x + dx, this.curPos.y + dy) |
|  | \*/ |
|  | ctx.fillRect(this.curPos.x + dx, this.curPos.y + dy, this.radius \* 1.5, this.radius \* 1.5); |
|  | } else { |
|  | /\* the default bubbleShape will be a circle |
|  | \* begin path |
|  | \*/ |
|  | ctx.beginPath(); |
|  | /\* To draw a circle filled with the current fillStyle we use |
|  | \* arc(x, y, radius, startAngle, endAngle, anticlockwise) |
|  | \* where (x, y) is the center point of the circle. |
|  | \* In this case, the radius is equal to this.radius with |
|  | \* center at coordinates (this.curPos.x + dx, this.curPos.y + dy) |
|  | \*/ |
|  | ctx.arc(this.curPos.x + dx, this.curPos.y + dy, this.radius, 0, Math.PI \* 2, true); |
|  | // fill path and end path |
|  | ctx.fill(); |
|  | } |
|  | }; |
|  | } |
|  |  |
|  | /\* function makeColor is used to convert an array of values |
|  | \* for example [196, 77, 55] into a color in HSL color model. |
|  | \* More about color models you can read at: |
|  | \* http://en.wikibooks.org/wiki/Color\_Models:\_RGB,\_HSV,\_HSL |
|  | \*/ |
|  | function makeColor(hslList, fade) { |
|  | var hue = hslList[0] /\*- 17.0 \* fade / 1000.0\*/ ; |
|  | var sat = hslList[1] /\*+ 81.0 \* fade / 1000.0\*/ ; |
|  | var lgt = hslList[2] /\*+ 58.0 \* fade / 1000.0\*/ ; |
|  | return "hsl(" + hue + "," + sat + "%," + lgt + "%)"; |
|  | } |
|  |  |
|  | // function phraseToHex is used to convert ASCII text into HEX coded text |
|  | function phraseToHex(phrase) { |
|  | // assign an empty string to hexphrase |
|  | var hexphrase = ""; |
|  | // for every character in the parameter phrase (...) |
|  | for (var i = 0; i < phrase.length; i++) { |
|  | // (...) add to hexphrase, the hexadecimal value of this character |
|  | hexphrase += phrase.charCodeAt(i).toString(16); |
|  | } |
|  | // return the converted string |
|  | return hexphrase; |
|  | } |
|  |  |
|  | /\* Function initEventListeners initializes the event listeners. |
|  | \* Event listeners allow us to create an interaction between the user and the website. They are special functions |
|  | \* that will launch defined by us code when a bounded event (for example click, pressed key, move of the mouse) |
|  | \* occurs. This allows us to give the user the ability to control animation with mouse / touch. |
|  | \*/ |
|  | function initEventListeners() { |
|  | /\* this statement will trigger the updateCanvasDimensions function [add\_line\_numbers] if our |
|  | \* page is resized by the user and will trigger the onMove function [add\_line\_numbers] when |
|  | \* the cursor is moved |
|  | \*/ |
|  | $(window).bind('resize', updateCanvasDimensions).bind('mousemove', onMove); |
|  |  |
|  | /\* this function will be triggered if the user touches the screen and moves their finger |
|  | \* (for example in smartphones) |
|  | \*/ |
|  | canvas.ontouchmove = function (e) { |
|  | // preventDefault statement terminates default action of the event [why do this?] |
|  | e.preventDefault(); |
|  | // trigger function onTouchMove [add\_line\_numbers] |
|  | onTouchMove(e); |
|  | }; |
|  | // this function will be triggered if the user touches the screen |
|  | canvas.ontouchstart = function (e) { |
|  | // preventDefault statement terminates default action of the event [why do this?] |
|  | e.preventDefault(); |
|  | }; |
|  | } |
|  |  |
|  | // function updateCanvasDimensions is used to control the size of the canvas |
|  | function updateCanvasDimensions() { |
|  | // basic variables, you can change them to resize the canvas element |
|  | canvas.attr({ |
|  | height: 500, |
|  | width: 1000 |
|  | }); |
|  | // assign to variables the values defined above |
|  | canvasWidth = canvas.width(); |
|  | canvasHeight = canvas.height(); |
|  | // trigger the draw function [add\_line\_numbers] |
|  | draw(); |
|  | } |
|  |  |
|  | // function onMove checks the position of the cursor and accordingly affects the animation |
|  | function onMove(e) { |
|  | // if pointCollection exists (...) |
|  | if (pointCollection) { |
|  | /\* (...) set the value of the mousePos property of pointCollection to the mouse coordinates |
|  | \* relative to the canvas element |
|  | \*/ |
|  | pointCollection.mousePos.set(e.pageX - canvas.offset().left, e.pageY - canvas.offset().top); |
|  | } |
|  | } |
|  |  |
|  | // function onTouchMove checks the position of a finger on the touch screen and accordingly affects the animation |
|  | function onTouchMove(e) { |
|  | // if pointCollection exists (...) |
|  | if (pointCollection) { |
|  | /\* (...) set the value of the mousePos property of pointCollection to the mouse coordinates |
|  | \* relative to the canvas element |
|  | \*/ |
|  | pointCollection.mousePos.set(e.targetTouches[0].pageX - canvas.offset().left, e.targetTouches[0].pageY - canvas.offset().top); |
|  | } |
|  | } |
|  |  |
|  | // function bounceName is used to repeatedly bounce our name |
|  | function bounceName() { |
|  | // trigger the shake function [add\_line\_numbers] |
|  | shake(); |
|  | // trigger again this function (bounceName) after 30 ms |
|  | setTimeout(bounceName, 30); |
|  | } |
|  |  |
|  | // [add] |
|  | function bounceBubbles() { |
|  | // trigger the draw function [add\_line\_numbers] |
|  | draw(); |
|  | // trigger the update function [add\_line\_numbers] |
|  | update(); |
|  | // trigger again this function (bounceBubbles) after 30 ms |
|  | setTimeout(bounceBubbles, 30); |
|  | } |
|  |  |
|  | // function draw is used to draw all points / bubbles forming the animation |
|  | function draw(reset) { |
|  | // assign to a local variable tmpCanvas our canvas (the element at index 0 of our canvas object) |
|  | var tmpCanvas = canvas.get(0); |
|  |  |
|  | // if the getContext property of our canvas is not defined (...) |
|  | if (tmpCanvas.getContext === null) { |
|  | // (...) end function |
|  | return; |
|  | } |
|  |  |
|  | // assign to variable ctx the context of the canvas element |
|  | ctx = tmpCanvas.getContext('2d'); |
|  |  |
|  | // the statement below is used to erase everything from the canvas element |
|  | ctx.clearRect(0, 0, canvasWidth, canvasHeight); |
|  |  |
|  | /\* the syntax below is an example of the ternary operator - shorthand for if ... else construction |
|  | \* if the shape of our bubbles is not defined use "circle" as the shape |
|  | \* otherwise use the current shape |
|  | \*/ |
|  | bubbleShape = typeof bubbleShape !== 'undefined' ? bubbleShape : "circle"; |
|  |  |
|  | // if pointCollection exists (...) |
|  | if (pointCollection) { |
|  | // (...) trigger the draw function of the pointCollection object [add\_line\_numbers] |
|  | pointCollection.draw(bubbleShape, reset); |
|  | } |
|  | } |
|  |  |
|  | function shake() { |
|  | // assign to a local variable tmpCanvas our canvas (the element at index 0 of our canvas object) |
|  | var tmpCanvas = canvas.get(0); |
|  |  |
|  | // if the getContext property of our canvas is not defined (...) |
|  | if (tmpCanvas.getContext === null) { |
|  | // (...) end function |
|  | return; |
|  | } |
|  |  |
|  | // assign to the variable ctx context of the canvas element |
|  | ctx = tmpCanvas.getContext('2d'); |
|  |  |
|  | // the statement below is used to erase everything from the canvas element |
|  | ctx.clearRect(0, 0, canvasWidth, canvasHeight); |
|  |  |
|  | /\* the syntax below is example of ternary operator - shorthand for if ... else construction |
|  | \* if the shape of our bubbles is not defined use "circle" as the shape |
|  | \* otherwise use current shape |
|  | \*/ |
|  | bubbleShape = typeof bubbleShape !== 'undefined' ? bubbleShape : "circle"; |
|  |  |
|  | // if pointCollection exists (...) |
|  | if (pointCollection) { |
|  | // (...) trigger the shake function of the pointCollection object [add\_line\_numbers] |
|  | pointCollection.shake(bubbleShape); |
|  | } |
|  | } |
|  |  |
|  | /\* function update is used to safely (only if the object exists) |
|  | \* update the pointCollection object |
|  | \*/ |
|  | function update() { |
|  | // if pointCollection exists (...) |
|  | if (pointCollection) |
|  | // (...) trigger the update method of the pointCollection object [add\_line\_numbers] |
|  | pointCollection.update(); |
|  | } |
|  |  |
|  | /\* function drawName is the main function in this script, it is used to |
|  | \* draw a string (parameter name) on the canvas in colors defined in the parameter |
|  | \* letterColors and to initialize the event listeners |
|  | \*/ |
|  | function drawName(name, letterColors) { |
|  | // trigger the updateCanvasDimensions function [add\_lines] |
|  | updateCanvasDimensions(); |
|  | // the variable g will store all of the points forming our animation, initial value is an empty array |
|  | var g = []; |
|  | // the variable offset will store the current width of our animation |
|  | var offset = 0; |
|  |  |
|  | /\* Function addLetter is used to retrieve data from alhpabet.js |
|  | \* for each given letter and transform them into Point objects. |
|  | \* Before reading comments to this function it is recommended to |
|  | \* read this article -> http://www.codecademy.com/forum\_questions/53385f2d52f8631f4200b18b |
|  | \* This is only the definition of the function, it will be triggered later in the code. |
|  | \*/ |
|  | function addLetter(cc\_hex, ix, letterCols) { |
|  | // if the variable passed as the letterCols parameter is defined (...) |
|  | if (typeof letterCols !== 'undefined') { |
|  | // (...) and is an array that stores array as a first element (...) |
|  | if (Object.prototype.toString.call(letterCols) === '[object Array]' && Object.prototype.toString.call(letterCols[0]) === '[object Array]') { |
|  | // (...) assign to variable letterColors the value passed as the parameter letterCols |
|  | letterColors = letterCols; |
|  | } |
|  | // (...) or if it is an array of numbers (...) |
|  | else if (Object.prototype.toString.call(letterCols) === '[object Array]' && typeof letterCols[0] === "number") { |
|  | // (...) assign to variable letterColors an array with one element inside - value of letterCols parameter |
|  | letterColors = [letterCols]; |
|  | } |
|  | } else { // if the variable passed as the letterCols parameter is not defined (...) |
|  | // (...) assign to variable letterColors an array with one element (dark gray color) array |
|  | letterColors = [[0, 0, 27]]; |
|  | } |
|  |  |
|  | // if the given letter (with hex code equal to the parameter cc\_hex) is defined in alphabet.js (...) |
|  | if (document.alphabet.hasOwnProperty(cc\_hex)) { |
|  | /\* Assign: |
|  | \* - to variable chr\_data: array of points defined in alphabet.js (property P) |
|  | \* - to variable bc: next color from letterColors array |
|  | \*/ |
|  | var chr\_data = document.alphabet[cc\_hex].P; |
|  | var bc = letterColors[ix % letterColors.length]; |
|  |  |
|  | // for every element of the chr\_data array (...) |
|  | for (var i = 0; i < chr\_data.length; ++i) { |
|  | // (...) assign to variable point the current element of the chr\_data array |
|  | point = chr\_data[i]; |
|  |  |
|  | /\* Add to array g a new Point object: |
|  | \* - 2d position of point is determined by values defined in alphabet.js, |
|  | \* to horizontal position (point[0]) we added offset. We have done that, |
|  | \* because every letter in alphabet.js is defined relatively to point (0, 0). |
|  | \* Without our addition all letters would be placed at a single stack. |
|  | \* - z coordinate is set to (0, 0), this is initial value. |
|  | \* - size of point is determined by value defined in alphabet.js |
|  | \* - color of point is a resultant of color bc (from letterColors array) with added |
|  | \* fade defined in alphabet.js. Fading is turned off by default, you can find |
|  | \* more information about fading in this article -> |
|  | \* http://www.codecademy.com/forum\_questions/5338606c282ae3de6c007ee3 |
|  | \*/ |
|  | g.push(new Point(point[0] + offset, |
|  | point[1], |
|  | 0.0, |
|  | point[2], |
|  | makeColor(bc, point[3]))); |
|  | } |
|  |  |
|  | // add to the variable offset width (property W) of the given letter (with hex code equal to parameter cc\_hex) |
|  | offset += document.alphabet[cc\_hex].W; |
|  | } |
|  | } |
|  |  |
|  | /\* Assign to the variable hexphrase the result of the phraseToHex function [add\_lines] called with passed |
|  | \* name as a parameter. As a result the variable hexphrase will store an array of letters which |
|  | \* creates our name coded in hex values. It is important, because letters in alphabet.js are |
|  | \* coded in this way. |
|  | \*/ |
|  | var hexphrase = phraseToHex(name); |
|  |  |
|  | // variable col\_ix stores index (in letterColors array) of next color to use |
|  | var col\_ix = 0; |
|  |  |
|  | /\* The for loop below is used to iterate through every letter in a hexphrase string. |
|  | \* But, previously to iterate through every element we incremented i by 1. In this case |
|  | \* we have to add 2 to i after every step because every letter in our array is represented |
|  | \* by two characters (for example A is depicted as 41). |
|  | \*/ |
|  | for (var i = 0; i < hexphrase.length; i += 2) { |
|  | /\* Variable cc\_hex stores the actual letter of hexphrase in the format used in alphabet.js |
|  | \* (A + hexvalue). Hexvalue is calculated by concatenating the character at index i with |
|  | \* the next character (index i + 1). |
|  | \*/ |
|  | var cc\_hex = "A" + hexphrase.charAt(i) + hexphrase.charAt(i + 1); |
|  | /\* Tigger the addLetter function [add\_lines] to add to our animation character |
|  | \* with hex value equal to cc\_hex and color defined in letterColors at index |
|  | \* col\_ix. This operation affects the value of variable g. |
|  | \*/ |
|  | addLetter(cc\_hex, col\_ix, letterColors); |
|  |  |
|  | /\* If statement below will increment col\_ix by 1 if current character |
|  | \* is not a space (hex value 20). We have to use this code to make sure |
|  | \* that spaces do not affect col\_ix (they are invisible, so we should not |
|  | \* 'reserve' color for them). |
|  | \*/ |
|  | if (cc\_hex != "A20") { |
|  | col\_ix++; |
|  | } |
|  | } |
|  |  |
|  | // for every element in array g (for every single point in our animation) (...) |
|  | for (var j = 0; j < g.length; j++) { |
|  | /\* Add to properties curPos.x and originalPos.x a left margin calculated from the formula: |
|  | \* width of our canvas / 2 - width of our animation / 2. We repeat this operation for |
|  | \* verical properties (curPos.y and originalPos.y), this time we affect the vertical position |
|  | \* but instead of using a variable we hard coded `105` as it is half of the height of our animation, why? |
|  | \* Because by checking values of property P[1] of every point in alphabet.js we can find out |
|  | \* that 210 is the difference between minimal and maximal vertical value. |
|  | \* In summary, code below is used to center your animation horizontally and vertically. |
|  | \*/ |
|  | g[j].curPos.x += (canvasWidth / 2 - offset / 2); |
|  | g[j].curPos.y += (canvasHeight / 2 - 105); |
|  | g[j].originalPos.x += (canvasWidth / 2 - offset / 2); |
|  | g[j].originalPos.y += (canvasHeight / 2 - 105); |
|  | } |
|  |  |
|  | // assign to variable pointColletion a new object of class PointCollection |
|  | pointCollection = new PointCollection(); |
|  | // set the points property of our newly created object to g, an array of points |
|  | pointCollection.points = g; |
|  | // trigger event handlers [add\_lines] |
|  | initEventListeners(); |
|  | } |
|  |  |
|  | /\* Set the reset property of the window object to false, this is |
|  | \* predefined value. This property is used only in the |
|  | \* draw method of the pointCollection object [add\_line\_numbers] |
|  | \*/ |
|  | window.reset = false; |
|  |  |
|  | // when the cursor leaves the site / document (...) |
|  | $(window).mouseleave(function () { |
|  | // (...) set the reset property of the window object to true |
|  | window.reset = true; |
|  | }); |
|  |  |
|  | // when the cursor enters the site / document (...) |
|  | $(window).mouseenter(function () { |
|  | // (...) set the reset property of the window object to false |
|  | window.reset = false; |
|  | }); |
|  |  |
|  | // assign to a variable the canvas element with id `myCanvas` |
|  | var canvas = $("#myCanvas"); |
|  |  |
|  | // declaration of the basic variables |
|  | var canvasHeight; |
|  | var canvasWidth; |
|  | var ctx; |
|  | var pointCollection; |
|  |  |
|  | // settings of our animation, you can try to change them, have fun :) |
|  | document.rotationForce = 0.0; |
|  | document.Friction = 0.85; |
|  | document.springStrength = 0.1; |
|  |  |
|  | /\* basic, predefined colors, used for example in exercise: |
|  | \* http://www.codecademy.com/courses/animate-your-name/2/3 |
|  | \*/ |
|  | var white = [0, 0, 100]; |
|  | var black = [0, 0, 27]; |
|  | var red = [0, 100, 63]; |
|  | var orange = [40, 100, 60]; |
|  | var green = [75, 100, 40]; |
|  | var blue = [196, 77, 55]; |
|  | var purple = [280, 50, 60]; |
|  |  |
|  | // this statement will trigger the updateCanvasDimensions function after 30 ms [add\_line\_numbers] |
|  | setTimeout(updateCanvasDimensions, 30); |