Week 7 WDD330

lunes, 29 de mayo de 2023 12:20

Ruth Cardona!

THIS KEYWORD (OOP)

THIS IS A KEYWORD WHOSE VALUES CHANGES DEPENDING ON HOW A FUCTION GETS CALLED

- TO ACCESS THE OBJECT, A METHOD CAN USE THE **THIS** KEYWORLD

```
let user = {
  name: "John",
  age: 30,

  sayHi() {
    // "this" is the "current object"
    alert(this.name);
  }

};

user.sayHi(); // John
```

ATTR
USER
METHOD

- IT'S VALUE IS EVALUATED AT CALL-TIME
- DOES NOT DEPEND ON WHERE THE METHOD WAS DECLARED BUT ON WHAT OBJECT IS BEFORE THE DOT.

- THIS KEYWORD CAN BE:

1. `this` in global context → OUTSIDE ANY FUCTION:
2. `this` in object construction
3. `this` in an object method
4. `this` in a simple function
5. `this` in an arrow function
6. `this` in an event listener

```
console.log(this) // Window
```
→ OBJECT

2. IN OBJECT CONSTRUCTION

```
function Person(age) {
  this.age = age
}

const greg = new Person(22)
const thomas = new Person(24)

console.log(greg) // this.age = 22
console.log(thomas) // this.age = 24
```

THIS REFERS TO THE INSTANCE

3- In an Object Method

```
function Human(name) {
  return {
    name,
    getName() {
      return this.name
    },
  }
}

const zell = new Human('Zell')
const vincy = new Human('Vincy')

console.log(zell.getName()) // Zell
```

THIS REFERS TO THE OBJECT

YOU CAN USE METHODS TO GET THE INSTANCE OF AN OBJECT.

4- In a simple function (and Anonymous)

```
function simpleFunction() {
  console.log(this)
}

const o = {
  sayThis() {
    simpleFunction()
  },
}

simpleFunction() // Window
o.sayThis() // Window
```

On browsers, "this" is always set to window in a simple function

5- In Arrow Fuction

```
const o = {
  doSomethingLater() {
    setTimeout(() => this.speakLeet(), 1000)
  },
  speakLeet() {
    console.log(`1337 15 4W350M3`)
  },
}
```

Using ARROW FUCTION WITHING AN OBJECT METHOD, THE "THIS" CONTEXT STAYS THE OBJECT, NOT WINDOWS.

A THIRD WAY TO CHANGE THE VALUE OF "THIS" WITHING ANY FUNCTION IS TO USE EITHER **BIND**, **CALL** OR **APPLY**.

- WHEN **THIS** IS REFERENCE IN AN ARROW FUNCTION, IT'S TAKEN FROM THE OUTER "NORMAL" FUNCTION

- ARROW FUNCTIONS DON'T HAVE THEIR OWN **THIS**

```
let user = {
  firstName: "Ilya",
  sayHi() {
    let arrow = () => alert(this.firstName);
    arrow();
  }
};

user.sayHi(); // Ilya
```

6. IN EVENT LISTENER

```
let button = document.querySelector('button')

button.addEventListener('click', function () {
  console.log(this) // button
})
```

- TO REMOVE AN EVENT LISTENER, THE CALLBACK PASS AS THE SECOND VALUE NEEDS TO BE A NAMED FUNCTION.

```
function someFunction() {
  console.log('do something')

  // Removes the event listener.
  document.removeEventListener('click', someFunction)
}

document.addEventListener('click', someFunction)
```

CSS-ANIMATIONS

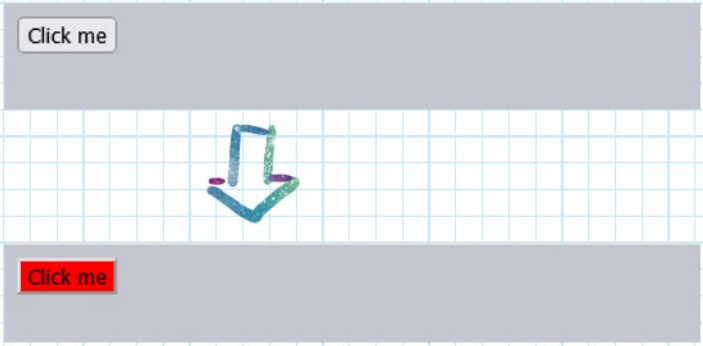
CSS TRANSITIONS:

- WE DESCRIBE A PROPERTY AND HOW ITS CHANGES SHOULD BE ANIMATED

```html
<button id="color">Click me</button>

<style>
  #color {
    transition-property: background-color;
    transition-duration: 3s;
  }
</style>

<script>
  color.onclick = function() {
    this.style.backgroundColor = 'red';
  };
</script>
```



- YOU CAN ANIMATE MULTIPLE PROPERTIES AT ONCE.

```css
#growing {
  transition: font-size 3s, color 2s;
}
```

PROPERTIES: 1. transition-property
2. transition-duration
4. transition-timing-function
3. transition-delay

1- TRANSITION-PROPERTY: A LIST OF PROPERTIES TO ANIMATE (LEFT, MARGIN-LEFT, HEIGHT, COLOR, ALL).

2- TRANSITION-DURATION: HOW LONG THE ANIMATION SHOULD TAKE (SECONDS, MILLISECONDS).

3- TRANSITION-DELAY: THE DELAY BEFORE THE ANIMATION (+ & - VALUES, SEC.)

- IF A TRANSITION-DELAY IS 1S AND TRANSITION-DURATION IS 2S THEN THE ANIMATION STARTS 1 SECOND AFTER THE PROPERTY CHANGE
  ↳ TOTAL DURATION: 2S

4- TRANSITION-TIMING-FUCTION: DESCRIBES HOW THE ANIMATION PROCESS IS DISTRIBUTED ALONG ITS TIMELINE (SLOW TO FAST OR VISUERSA)

- IT'S ACCEPTS TWO VALUES:
  - BEZIER CURVE
  - STEPS

BEZIER CURVE IT HAS 4 CONTROL POINTS

X Y

1. First control point: `(0,0)`.
2. Last control point: `(1,1)`.
3. For intermediate points, the values of `x` must be in the interval `0..1`, `y` can be anything.

SYNTAX: CUBIC-BEZIER (X2, Y2, X3, Y3)

, WE ONLY SET 2 & 3 CONTROL POINTS

- THE TIMING FUNCTION DESCRIBES HOW FAST THE ANIMATION PROCESS GOES

- The `x` axis is the time: `0` – the start, `1` – the end of `transition-duration`.
- The `y` axis specifies the completion of the process: `0` – the starting value of the property, `1` – the final value.

EXAMPLE

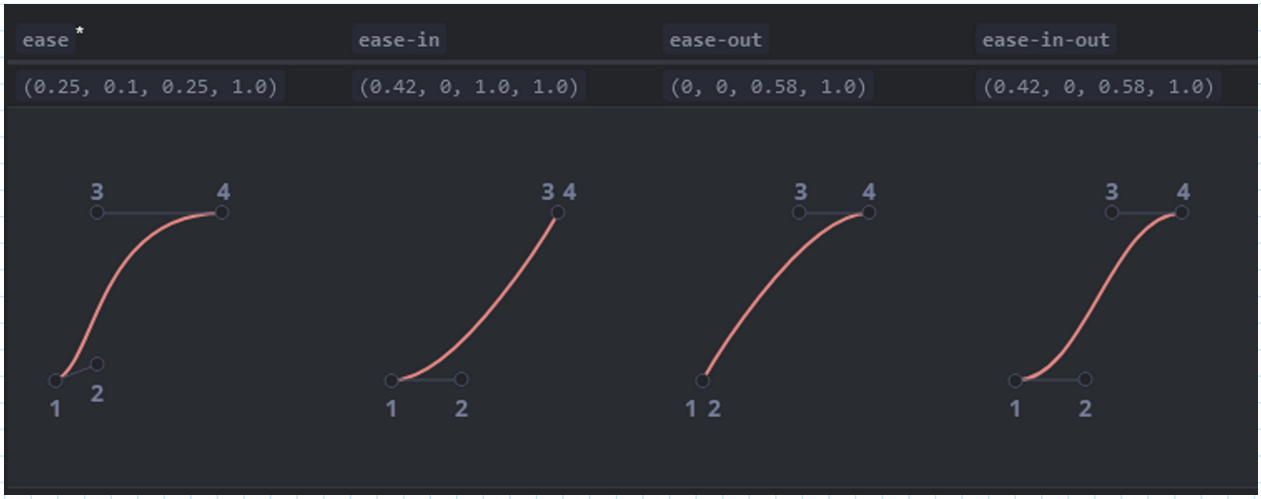
The simplest variant is when the animation goes uniformly, with the same linear speed. That can be specified by the curve `cubic-bezier(0, 0, 1, 1)`.

Here's how that curve looks:

`cubic-bezier(0.0, 0.5, 0.5 ,1.0)`.

The graph:

BUILD-IN CURVES: LINEAR, EASE (→ DEFAULT), EASE-IN, EASE-OUT, EASE-IN-OUT



BEZIER CURVE CAN MAKE THE ANIMATION EXCEED ITS RANGE

STEPS `steps(number of steps[, start/end])`

ALLOWS SPLITTING A TRANSITION INTO MULTIPLE STEPS.

```css
#stripe.animate {
  transform: translate(-90%);
  transition-property: transform;
  transition-duration: 9s;
  transition-timing-function: steps(9, start);
}
```

NUM OF STEPS

MEANS THAT IN THE BEGINNING OF ANIMATION WE NEED TO MAKE THE FIRST STEP IMEDIATLY

IT CAN BE START OR END

The process is progressing like this:

- `0s` – `-10%` (first change in the beginning of the 1st second, immediately)
- `1s` – `-20%`
- ...
- `8s` – `-90%`
- (the last second shows the final value).

So the process for `steps(9, end)` would go like this:

- `0s` – `0` (during the first second nothing changes)
- `1s` – `-10%` (first change at the end of the 1st second)
- `2s` – `-20%`
- ...
- `9s` – `-90%`

EVENT: "TRANSITIONEND"

IT TRIGGERS WHEN THE CSS ANIMATION FINISHES

- IS USED TO DO AN ACTION AFTER THE ANIMATION IS DONE

```js
boat.onclick = function() {
  //...
  let times = 1;

  function go() {
    if (times % 2) {
      // sail to the right
      boat.classList.remove('back');
      boat.style.marginLeft = 100 * times + 200 + 'px';
    } else {
      // sail to the left
      boat.classList.add('back');
      boat.style.marginLeft = 100 * times - 200 + 'px';
    }

  }

  go();

  boat.addEventListener('transitionend', function() {
    times++;
    go();
  });
};
```

PROPERTIES:

- EVENT.PROPERTYNAME = THE PROPERTY THAT HAS FINISHED ANIMATING
- EVENT.ELAPSEDTIME = THE TIME (SECONDS) THAT THE ANIMATION TOOK WITHOUT TRANSITION-DELAY

KEYFRAMES: @KEYFRAMES

TO JOIN MULTIPLE SIMPLE ANIMATIONS TOGETHER

```css
@keyframes go-left-right {        /* give it a name: "go-left-right" */
  from { left: 0px; }             /* animate from left: 0px */
  to { left: calc(100% - 50px); } /* animate to left: 100%-50px */
}

.progress {
  animation: go-left-right 3s infinite alternate;
  /* apply the animation "go-left-right" to the element
     duration 3 seconds
     number of times: infinite
     alternate direction every time
  */
```

PERFORMANCE:

WHEN THERE IS A STYLE CHANGE, THE BROWSER GOES THROUGH 3 STEPS:

1. LAYOUT: RE-COMPUTE THE GEOMETRY AND POSITION OF EACH ELEMENT
2. PAINT: RE-COMPUTE HOW EVERYTHING SHOULD LOOK LIKE AT THEIR PLACES INCLUDING BACKGROUND COLORS

3. COMPOSITE: RENDER THE FINAL RESULT INTO PIXELS ON SCREEN, APPLY CSS TRANSFORMS IF THEY EXIST.

- DURING CSS ANIMATIONS, THIS PROCESS REPEATS EVERY FRAME.
- ANIMATIONS THAT SKIP THE LAYOUT STEP ARE FASTER AND BETTER IF PAINT IS SKIPPED TOO

> The `transform` property is a great choice, because:
>
> - CSS transforms affect the target element box as a whole (rotate, flip, stretch, shift it).
> - CSS transforms never affect neighbour elements.

- THE BROWSER APPLY TRANSFORM "ON TOP" OF EXISTING LAYOUT AND PAINT CALCULATION, IN THE COMPOSITE STAGE.
- BY USING TRANSFORMS ON AN ELEMENT, YOU COULD ROTATE AND FLIP IT, STRETCH AND SHRINK IT, MOVE IT AROUND, ETC.
  - ↳ TRANSLATEX(...)
  - ↳ TRANSFORM: SCALE