how to install webpack ?

→ to use npm run build to copy from index.htm in src to index.html in dist

→ you import all your .mjs files in src, to index.js in src, this will bundle them all into one js file, bundle.js.

→ you only import bundle.js in your js file.

→ make sure your entry is src/index.js → so that webpack knows where to start

ui.js

→ you update your minute display, second display

→ you get the remaining time in seconds.

→ ex: 90 seconds → 01:30

floor (90/60) = 1.5 = 1 (place in minutes).

90 % 60 = 1 → reminder = 30 so place 30.

→ you will get time from timer.js.

→ String (any number) will give you the number in a string.

→ string (7) = "7"

→ string.padStart (number of min digits, what to put at the begining when the length is less). (→ you can put anything)

padStart (2, "0") when string = "AB" → nothing

" " = "A" → "0A"

padStart (3, "0") → string = "A" → "00A"

updateUI → updateTimer.

switch mode → changeColor.

→ document.querySelector ("name").style.name = "value"

↳ will change style

→ condition ? true : false

↳ Ternary operator

start / pause / reset are all timer.js function.

you call them in ui.js.

→ use arrow function so that we don't call them instantly.

→ one of the event listeners is DOMContentLoaded

↳ Run when all UI is loaded into page.

→ document.addEventListener (event, function)

DOMContentLoaded ← click (anywhere in the page)

hover →

↳ can only apply to document and it means once all of your document elements are loaded. do the function

→ once the page is loaded get the preffered work durations from localstorage, and place them in inputs, also send them to the timer.

input.addEventListener("change", () => {
↳ when the user changes the input.
→ when it changes → change localStorage
→ change timer.js logic

→ change vs input
. Both mean that user change value.
. But input runs instantly.
. change runs when user leaves input field.

→ element.textContent = "value"
↳ will change the value to that.

→ input.value will change input value.

→ when you have a pointer to element ex btn. and you make a variable = to that. You can now modify and work with the element using variable
→ you can pass it through modules easily.

notification.js

→ you put a sound.mp3 in your files.
→ sound has a class called Audio.
→ Sound.play() will play it.

→ const sound = new Audio("./Audio.mp3") → works normally without bundlers.
→ sound.play to play.
→ when using bundlers like webpack, the actual working code will be in dist/
→ So they expect it to be there, but actually it is not.
→ So you use require("mp3")
↳ This will find the audio.

→ const sound = new audio (require("./mp3"));
sound.play();
↳ will put the right path
(use for assets) → when they are in src and you want them in dist/.

→ Notice that when you npm run build all your assets will be loaded into dist/ manually.

settings.js

→ localStorage.setItem("key", value)
will store the value as string.
localStorage.getItem("key") will return the value as string.

Timer.js

→ you need a variable, when you click start, it become true. → if time was not running and you click start it becomes true
→ That value will prevent multiple timers from going at once, which may cause some faster count down.
→ you need start timer, pause timer, Reset timer functions.
→ you need to know how many works sessions you did, for long break implementation.

setTimeout vs setInterval
→ both have the same syntax (function, time in ms)
→ Timeout will run once, after the time specified for it.
→ setInterval will run every (after time you put).
→ ex: time = 1000
Timeout → will run once every 1s, setInterval will run every second.

→ you can stop the interval from going, by using clearInterval (interval_name)

So let timer = setInterval (function, time in ms)

after you want to stop it (calling the function) use clearInterval (timer)

→ So you need a variable to store your setInterval

→ when you click start, while time is running do nothing.

→ every second: timeleft >0 ? timeleft --;

→ update UI with time.

→ Time passed in seconds. (and will be displayed)

→ Time left =0 then you:

→ switch modes

→ update remaining time

→ clearInterval → stop clock from going

→ playsound → automatically played.

→ Increase work session numbers.

↳ will help with the long break

initial work = 25

break = 5

long = 25

switch durations (long, short, work)

workduration = work * 25

⋮

long

short

→ To change between modes:

1- add buttons

2- add eventListeners and call functions from timer.js

3- pass the type of session you want.

4- pass update UI, to update timers.

5- pass switchmode to add colors.

6- make if statements for each case

→ work → short → long.

. in all you stop the interval

. you make time left = user preference

→ For pause, you just make running = false.

→ clearInterval

→ For reset you work depending on the mode.

→ clearInterval.

→ update UI

→ running false

→ You can export functions

→ But you can't export variables even when they are let

→ To export them you use an object and put the variables in.