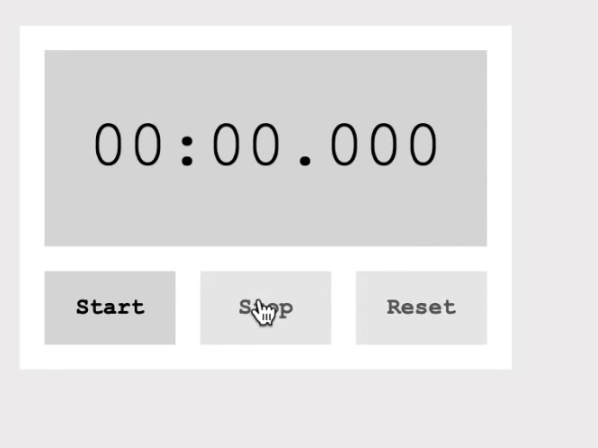
ストップウォッチを作る

目標物



タイマーのマークアップ、ボタンのマークアップ、要素の取得

まずは最小限のマークアップで動作するものを作っていく。

Divタグでtimerをbuttonタグで開始、停止、リセットを作る

<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Stopwatch</title>

<link rel="stylesheet" href="css/styles.css">

</head>

<body>

<div id="timer">00:00.000</div>

<button id="start">Start</button>

<button id="stop">Stop</button>

<button id="reset">Reset</button>

<script src ="js/main.js"></script>

</body>

</html>

必要な要素をすべて取得しておく

////

'use strict'

{

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

}



///

全体をブロック { } で囲っていますが、  
囲ったほうがいいというのはなぜでしょうか？

 先生

変数の有効範囲をスコープと呼び，変数はそのスコープの中でのみアクセスすることができます．  
constやletで宣言した変数のスコープはその変数を囲む一番内側のブロック（{}）と決まっています．

一般的に変数のスコープはできるだけ小さくした方がよいとされています．なぜかというと，スコープが広すぎると変数同士の名前が衝突したり思いもよらないところで変数の値が変更されてしまったりすることが起き，開発を進めるのが難しくなるからです．

特に，すべてのブロックの外側（'use strict;'のスコープです）はグローバルスコープと呼ばれ，あらゆるJavaScriptコードから参照できる一番広いスコープとなっており，ここに変数を配置するのは先述の理由から良い習慣ではないと考えられています．

ですので，このレッスンでやったように{}でブロックを作ることで変数のスコープを最小限に収めています．

カウントアップ機能の実装（clickイベントの設定、Date.now()、setTimeout()）

Startボタンにclickイベントをつける

現在時刻をstartTimeという変数で取得。基準となる日時からの経過ミリ秒を表示すれば良いのでDate.now()としておけば良い。startTimeはaddEventListnerのブロック外でも使うので外で宣言しておく。再代入するのでletとしておく。

カウントアップの処理はcountUp()関数を実行することとする。関数も定義する。startTimeからの経過時間を表示する。（現在時刻からstartTimeを引くと表示できる）

setTimeout()を使って10ミリ秒後にcountUp()自身を呼び出す。

'use strict'

{

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

let startTime;

function countUp() {

console.log(Date.now() - startTime);

setTimeout(() => {

countUp();

}, 10);

}

start.addEventListener('click', () => {

startTime = Date.now();

countUp();

});

}

////

setTimeout(countUp,10);

ではなく

setTimeout(()=>{

countUp();

}, 10);

となっているのは何故ですか。

 先生

こちらはどちらでもいいですね。

ただ、

setTimeout(()=>{

countUp();

}, 10);

と書いておくと、

* setTimeout()に渡したい処理をあとから増やすことができる
* countUp()に（必要だったら）引数を渡すことができる

という利点がありますね。

将来的にそういった可能性があることも考慮して、こういった書き方もすることもあります。参考にしてみてください。

補足：setTimeoutやsetIntervalには関数()は付けてはいけないようだ！そのためsetTimeout(countUp, 10)では引数を渡すことはできない。

////

"setInterval"と"setTimeout"の動画では「関数を引数として渡す場合、（）の有無で処理の内容の意味合いが変わってくる」との解説でしたが、何故今回はどちらでも良いのでしょうか？？

良いご質問ですね。

ここで、

setTimeout(() => {

countUp();

}, 10);

としていますが、これを一行で書くと、

setTimeout(() => { countUp(); }, 10);

となります。

ここで、setTimeou()はどういう構造になっているかというと、

setTimeout(関数, 10);

ですね。

では、この「関数」に()がついているかというと**ついていません**（ここで渡している「関数」は(() => { countUp(); }の無名関数で、最後に()はついていません）。

この「関数」のなかで実行するcountUp()には()がついていますが、渡す「関数」自体には()がついていないので、setTimeout()やsetInterval()にわたす関数には()をつけてはいけない、という説明とは矛盾していないと思うのですがいかがでしょうか。

////

ブロック内の「 startTime = Date.now(); 」にはなぜ変数の宣言letがいらないのでしょうか？ブロックの外の「let startTime ;」ですでに宣言が終わっているという解釈で良いのでしょうか？

 先生

変数startTimeをブロックの外で宣言しましたが、なぜ以下のようにしないんでしょうか？

let startTime = Date.now();  
添付のようにコードを書いても問題なく動きました。

startTimeの宣言時にDate.now();で初期化してしまうと、main.jsが読み込まれたときの時刻がstartTimeに代入されます。

今回startTimeに代入したい時刻はStartボタンがクリックされたときの時刻なので、start.addEventListener() の中で初期化する必要があります。

一見問題なく動作しているようにも見えるのですが、更新速度が早くStartボタンを押したときの経過時刻が見えないので、試しにsetTimeout()のタイムアウト時間を1000にしてみると0からスタートしないことがわかるかと思います。

またブロック内の「 startTime = Date.now(); 」にはなぜ変数の宣言letがいらないのでしょうか？  
ブロックの外の「let startTime ;」ですでに宣言が終わっているという解釈で良いのでしょうか？

そうですね、その理解で問題ありません。

////

Date.now()とnew Date()の違いがわかりません

2020年5月18日

 ユーザー

「[詳解JavaScript オブジェクト編／#21 現在日時を扱ってみよう](https://dotinstall.com/lessons/basic_javascript_objects_v2/52321)」での、

* 現在日時を表す値は new Date() で作ることができる〜

という内容を思い出し、本動画内のstartTime = Date.now();にもnew Date()を使用したところ、動作自体に問題はありませんでした（同様にconsole.log()の箇所にもnew Date()を使用しました。）

それぞれの違いがわからず、どのような理由でここではDate.now()を利用されているのかご教示いただけますでしょうか？

* [main.js](https://dotinstall.com/questions/320yqa0/files/19142" \t "_blank)

 先生

まずはそれぞれの定義を見てみましょう

Date()  
新たな Date オブジェクトを生成します。

Date.now()  
現在の時刻に対応する数値、すなわち UTC の 1970 年 1 月 1 日 00:00:00 から経過したミリ秒 (閏秒は無視) を表す数値を返します。

<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Date>

一方はオブジェクトを生成し、もう一方は数値を返してくれます。

ですが、Dateオブジェクトの引き算がその差分を返してくれているのですね。  
そのため今回のような使い方の場合、違いがないように見えています。

本レッスンでは秒数（数値）を表示させたいので、Dateオブジェクトは不要で、最初から秒数（数値）を扱うDate.now()を使っています。

Dateオブジェクトをそのまま使いたい場合にはnew Date()を使うと良いかなと思います。

 ユーザー

ご回答いただきありがとうございます！

おかげさまで「オブジェクトを生成する」「数値を返す」というそれぞれの違いと、「最初から秒数を扱う」ためにここではDate.nowを使用している点についても理解できました。

しかしながら、もう少し補足をお願いできますでしょうか？

Dateオブジェクトの引き算がその差分を返してくれている

* このご説明は(new Date() - startTime)の箇所のことで合っていますか？
* また、ご説明の通り解釈しますと、「オブジェクト - オブジェクト（startTimeはnew Date()を代入しているので） = 数値」という解釈で合っていますでしょうか？
* また、こういったデータ型の変化（？）は「こういうもの」と理解するしかないものでしょうか？

データ型の理解がなかなか難しく…お手数ですが改めてご説明をお願いいたします。

 先生

このご説明は(new Date() - startTime)の箇所のことで合っていますか？

そうですね。ここの引き算のことです

また、ご説明の通り解釈しますと、「オブジェクト - オブジェクト（startTimeはnew Date()を代入しているので） = 数値」という解釈で合っていますでしょうか？

Dateオブジェクトどうしの引き算は数値が返ってくるという解釈で大丈夫です。

一般的にオブジェクトどうしの+や-は成り立たないことの方が多いので特殊ケースだと思っていた方が良いかなと思います。

また、こういったデータ型の変化（？）は「こういうもの」と理解するしかないものでしょうか？

オブジェクトのメソッドはどんな値（オブジェクトなのか数値なのか文字列なのか）を返すのかというのは慣れてくるとなんとなく予想が付いてきますが、最初のうちはつどリファレンスを参照して確認しながら実装するのがおすすめです

////

# コンソールの結果が10ミリ秒間隔にならないのはなぜですか？

2020年6月4日

 ユーザー

コンソールで動作確認をする場面について、質問があります。  
  
例えば、動画の2:41辺りで、  
  
2423  
2435  
2447  
2457  
  
上記のようにコンソールでは表示されていましたが、  
  
2457 - 2447 = 10 （ミリ秒差）  
2447 - 2435 = 12 （ミリ秒差）  
2435 - 2423 = 12 （ミリ秒差）  
  
と、上記のようにそれぞれの表示ミリ秒の間隔を計算すると、１０ミリ秒毎に処理を施すコードを書いたにも関わらず、１０ミリ秒毎になっていません。これは、他の場面でも同じで、他の場面では、１１ミリ秒差というのも見当たりました。  
  
これは何故なのでしょうか？  
  
ご教示のほど、よろしくお願いいたします。

 先生

こんにちは。ご質問ありがとうございました。

上記のようにそれぞれの表示ミリ秒の間隔を計算すると、１０ミリ秒毎に処理を施すコードを書いたにも関わらず、１０ミリ秒毎になっていません。これは、他の場面でも同じで、他の場面では、１１ミリ秒差というのも見当たりました。

良いご質問ですね。

これはPCが1ミリ秒ごとに処理をしているわけではない、という理由です。下記のMDNでも説明があります。

<https://developer.mozilla.org/ja/docs/Web/API/WindowOrWorkerGlobalScope/setTimeout>  
※ 「遅延が指定値より長い理由」の項です

こちらによるとHTML5では4ミリ秒が最低の単位で、それ以下の単位を指定したとしても正確には処理されないことがわかります。

以上の理由により、十分に小さいミリ秒では数ミリ秒単位で誤差が出る、ということですね。なお、PCの負荷が高まっている場合はもっと誤差がでる場合もあります。

PCのCPUは負荷が高まってもなんとか処理を続行しようとするので、こうした仕様になっているのだな、と理解しておくといいでしょう。

////

countUp内でsetTimeoutとすることでcountUpからcountUpを呼び出しています。  
そうすることで無限に処理が連鎖しているのがレッスンのコードです。

setTimeoutをcountUpの外に出してしまいますと、countUpは一度しか実行されません。  
それが console に一度しか出力されない理由となります。

タイマーを表示させる（カウントアップされる値を秒や分に変換し、桁数を固定した上で画面に表示させる）newDat(), padStart(),string()

まずはdateオブジェクトを作って、そのメソッドを使うのが良さそう。New Dateとして先の引き算をそのまま入れる。これをdに代入しメソッドを使う。

Timer.textContentに文字列をテンプレートリテラルで表示させれば良い。

ダミーで入れたように桁数を2桁、2桁、3桁としたい。padStart()を使う。（これは文字列にしか使えないためstring()を使用する）

'use strict'

{

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

let startTime;

function countUp() {

// console.log(Date.now() - startTime);

const d = new Date(Date.now() - startTime);

// const m = d.getMinutes();

// const s = d.getSeconds();

// const ms = d.getMilliseconds();

const m = String(d.getMinutes()).padStart(2, '0');

const s = String(d.getSeconds()).padStart(2, '0');

const ms = String(d.getMilliseconds()).padStart(3,'0');

timer.textContent = `${m}:${s}.${ms}`;

setTimeout(() => {

countUp();

}, 10);

}

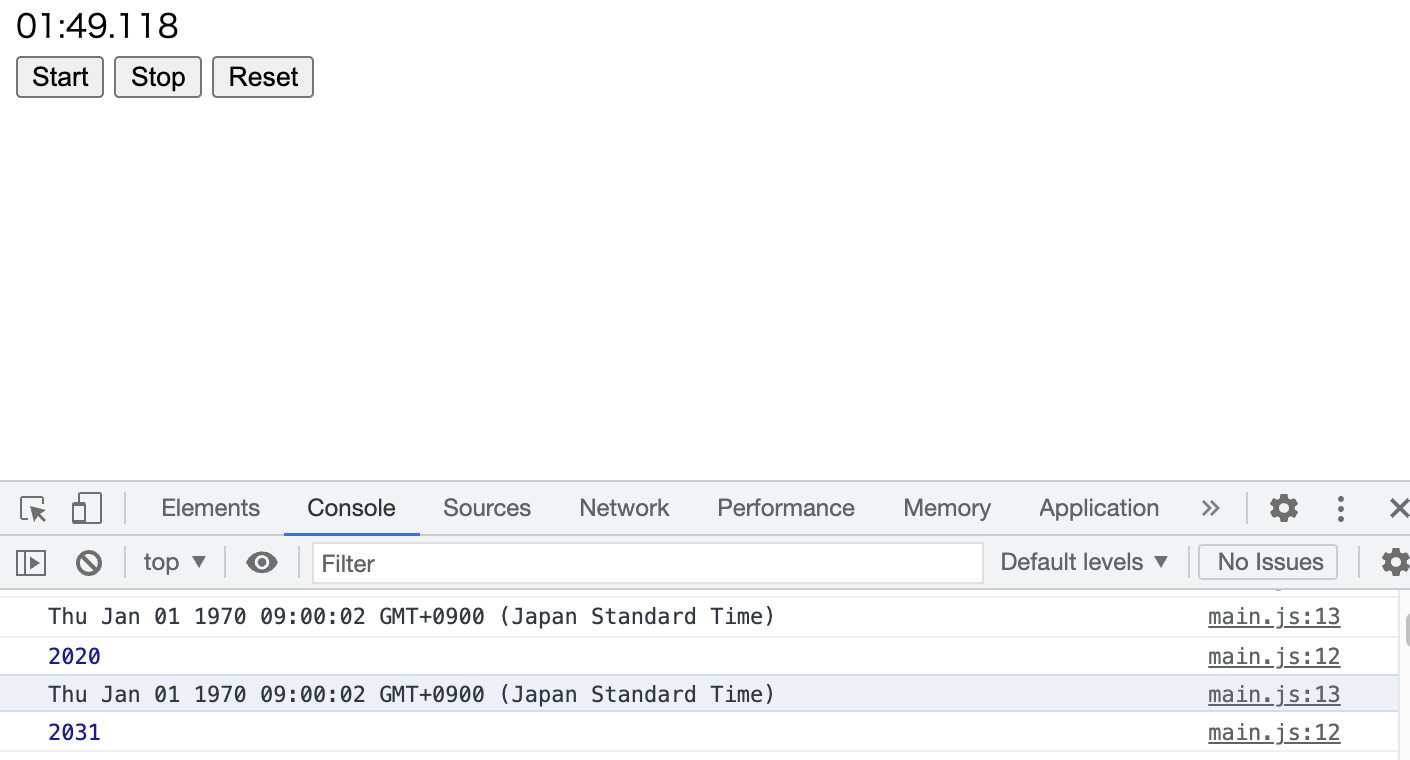
start.addEventListener('click', () => {

startTime = Date.now();

countUp();

});

}



console.log(Date.now() - startTime);

console.log(new Date(Date.now() - startTime));

これらを表示させてみた。

///



///

画像12行目ではnew演算子を使いDateオブジェクトをインスタンス化していますが、なぜ24行目ではnewを使わなくてもDateをインスタンス化出来てるのでしょうか？

これはDate.now()が、インスタンス化しなくても呼び出せるメソッド（いわゆる静的メソッド）として定義されているからですね。

取得できる値は、

* new Date().getTime()

としても

* Date.now()

としても同じですが、後者は現在時刻の値しか取得できないけれども短く書ける、というメリットがあります。

どちらも使えるようにしておくといいでしょう。

それからnewせずにクラスから直接呼び出せる静的メソッドについては以下も参考にしてみてください。

・#33 静的メソッドを使ってみよう | 詳解JavaScript オブジェクト編  
<https://dotinstall.com/lessons/basic_javascript_objects_v2/52333>

////

# setTimeout(countup(), 10); だとエラーになるのはなぜですか？

2020年4月26日

 ユーザー

setTimeoutの中身にcountup()をアロー関数で入れないといけないのはなぜでしょうか。

 先生

setTioumoutの第一引数は、タイマーが満了した後に実行したい 関数を指定する必要があります。

setTimeout(countup(), 10);の書き方ですと、countup()の戻り値をsetTimeoutの引数に指定していることになりますので、  
Javascriptの構文として正しくないためエラーとなってしまいます。

////

const d = new Date(Date.now() - startTime);  
は何を行っているのでしょうか。  
new Dateで現在の時刻を取得しさらに()の中にDate.now() - startTimeを何のために記入しているのですか？

 先生

まずはDate.now()は1970年から現在までのミリ秒を返します。（なぜ1970年かは下のリンクを参照ください）

<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Date/now>

ストップウォッチをスタートした時にそれがstartTimeに保存されます。

そしてたとえば10秒後にDate.now()とすると先ほどのstartTimeから10000ミリ秒足された値になっているはずですよね。なのでDate.now() - startTimeとすると10000が取り出されます。

new Date(..)の..の部分は何も指定していないと現在時刻になるのですが実はミリ秒も指定することができて、その場合は1970年からのミリ秒として扱われます。

それをどうしてdに入れているかと言いますと、そこから秒や分を取り出すのが非常に簡単になるからです。自分で60や1000で割ったりしても計算できますが、このようにnew Dateに代入して取り出すのはよく使うテクニックなので覚えて起きましょう。

ということでDate.now()は1970年から現在までのミリ秒を返すこと、new Date()の引数はそれを指定できるとことを押さえておいてください。

またご不明なことがあればいつでもご質問くださいね。一緒に頑張りましょう！

 ユーザー

回答ありがとうございます。重ねてすみません。  
startTimeはクリックをした時間で止まるので、例えばそこから10秒後には  
Date.now()-startTimeの値が10000となり  
new Date(10000)となる。  
これは1970年から10000ミリ秒後という意味で10000という数値が出力され  
この数値をタイマーとして利用しているということでしょうか？

 先生

はい、ご理解の通りでございます！

1970年からというのが違和感があるかと思いますが、実際には日時は使わないのでこの様に使うテクニックというわけです。

////

0:11　色々な方法がありますが、まずはこちらの経過時間で Date オブジェクトを作ってあげて、そのメソッドを使ってあげるのが簡単でしょう。  
  
とありますが、この説明のコードの内容がいまいちわかりませんでした。  
  
<https://dotinstall.com/lessons/basic_javascript_objects_v2/52330>  
の動画のイメージがあって、newを使うにもDate()クラスがないのでは？と思ってしまいます。  
勉強不足で大変申し訳ありませんが、ご教授していただけると幸いです。

 先生

こんにちは。

Date クラスが突然出てきて利用できるのは、確かに不思議ですね。

実は、 JavaScript には組み込みのオブジェクトがいくつかあって、クラスを定義しなくてもそのまま使うことができます。

そのうちの一つが Date オブジェクトです。  
一般的なビルトイン関数やクラスは以下になります。

<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects>

また、以下リンクに Date クラスが持っているメソッドなどの説明があります。

<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Date>

 ユーザー

回答ありがとうございます。  
元々javascriptにはDateオブジェクトがあって、newを使ってインスタンス、メソッドを使用しているという解釈をしました。

////

# new Date()に渡すのは年月ではないのですか？

Dateのコンストラクターには以下の4種類があります。

1. 引数なし
2. 整数値の引数一つ
3. 日付を表す文字列の引数一つ
4. 年、月インデックス、日、時、分、秒、ミリ秒を引数にとる。ただし日、時、分、秒、ミリ秒は省略可

「new Date()を習ったオブジェクト編の授業」で扱われているものはこの4番目のコンストラクタに当たりまして、一方、今回のコンストラクタは2番目のものになります。

この2番目の整数を一つ引数にとるコンストラクタは、UTC時間で1970年1月1日00:00:00に引数に与えられた数値をミリ秒として加えた日時を返すコンストラクタになっております。

///

# constを使って値が再代入できるのはなぜですか？

 先生

ご利用ありがとうございます。

constは、変数のスコープ内で再代入できないという解釈がより正しい解釈となります。  
今回の例で言いますとcountUp関数から抜けてしまうことで変数dなどのスコープから外れてしまいます。

よって、再度countUpを呼び出した場合には再度生成されますので今回の場合は問題なく  
処理されます。

///

countUp は処理の内容の定義を保持しているのに対し， countUp() とすると定義されている処理を実際に実行します。  
まず setTimeout は引数として処理の内容の定義を受け取るので， setTimeout(countUp, 10); となります。  
setTimeout(countUp(), 10); としてしまうと setTimeout に対して処理の内容の定義ではなく，処理を実際に実行した結果が渡されてしまうのでうまく動作しません。

同様に setTimeout(() => { ... }, 10); とした場合は () => { ... } が処理の内容の定義となります。  
() => { ... } 内では実際に処理を実行する必要があるので， countUp() とします。

///

停止、リセットの機能を実装する。clickEvent(), clearTimeout()

Stopボタンでカウントアップを停止する。setTimeoutをキャンセルすれば良いのでclearTimeout()が使える。そのためにはtimeOut用のIDが必要になるが、setTimeout()の返り値で得ることが可能。timeoutIdをclearTimeout()に渡してあげれば次のsetTimeout()がキャンセルされる。

Resetはtimerを元の表記に戻してやれば良い。今回はtimer.textContentに00:00.000を入力してやる。

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

let startTime;

let timeoutId;

function countUp() {

// console.log(Date.now() - startTime);

const d = new Date(Date.now() - startTime);

const m = String(d.getMinutes()).padStart(2, '0');

const s = String(d.getSeconds()).padStart(2, '0');

const ms = String(d.getMilliseconds()).padStart(3,'0');

timer.textContent = `${m}:${s}.${ms}`;

timeoutId = setTimeout(() => {

countUp();

}, 10);

}

start.addEventListener('click', () => {

startTime = Date.now();

countUp();

});

stop.addEventListener('click', () => {

clearTimeout(timeoutId);

});

reset.addEventListener('click', () => {

timer.textContent = '00:00.000';

});

# Startの次にResetはなぜ押せないのでしょうか？

2020年5月3日

 ユーザー

まだsetbuttonStateInitialなどを入力していないこの状態でもStartボタンの後はStopボタンしか機能しません。スタート→ストップ→リセットではなくスタートの次に直接リセットを押すことはなぜ出来ないのでしょうか。（クリックできてもカウントがリセットされない。）  
後に習うclearTimeoutでカウントをキャンセルしなくても、この動画の最終ソースコードの状態でリセットボタンを押すことで「00:00.000」に戻すことは可能ではないでしょうか。。  
  
簡単な内容でしたら申し訳ございません。  
お答えいただけると幸いです。よろしくお願いします。

 先生

勉強お疲れ様です．質問ありがとうございます．

まだsetbuttonStateInitialなどを入力していないこの状態でもStartボタンの後はStopボタンしか機能しません。スタート→ストップ→リセットではなくスタートの次に直接リセットを押すことはなぜ出来ないのでしょうか。（クリックできてもカウントがリセットされない。）

実際にはResetボタンを押すことができて機能もしています．  
Resetボタンの挙動を確認すると以下のようになっています．

reset.addEventListener('click', () => {

timer.textContent = '00:00.000';

});

ここでは表示テキストを '00:00.000' に書き換えることによってリセットを実現しています．  
しかし，Startボタンを押して countUp 関数が走っている状態で表示テキストを書き換えても， countUp 関数内の

timer.textContent = `${m}:${s}.${ms}`;

ですぐに上書きされてしまいます．  
これがResetが機能していないように見える原因です．

後に習うclearTimeoutでカウントをキャンセルしなくても、この動画の最終ソースコードの状態でリセットボタンを押すことで「00:00.000」に戻すことは可能ではないでしょうか。。

今回のコードでは「Resetボタンはタイマーが動いていない状態で押される」という制約をもとに設計していますが，もちろんStartボタンを押してタイマーが動いている最中に時間のカウントをリセットしてしまうこともできます．

そもそも経過時間をどのように算出しているかに着目すると， countUp 関数内で

const d = new Date(Date.now() - startTime);

のようにして「現在時刻 - 開始時刻」として算出していることがわかるかと思います．  
経過時間をリセットするためにはこのうち開始時刻のほうをセットしなおせばいいので

reset.addEventListener('click', () => {

timer.textContent = '00:00.000';

startTime = Date.now();

});

のようにすればよいことがわかるかと思います．(これだけではダメだと思う。clearTimeoutしなければならないだろう)

////

# timer.textContentはtimer.valueではだめですか？

2020年7月14日

 ユーザー

timer.textContent = `${m}:${s}.${ms}`

というコードがありますが、こちらについて

timer.value= `${m}:${s}.${ms}`

でも動くかと思いましたがが、動きませんでした。理由は何になりますでしょうか？いまいち.textContentと.valueの違いを理解できておりません。

 先生

textContentとvalueの違いですね。textContentはHTMLのタグに囲まれたテキスト、valueはフォーム部品（inputタグなど）が保持している値を指します。

例えば

<p>ここがtextConentです</p>

<input type="text" value="ここがvalueです" >

という具合です。

今回のtmerはdivで入れるテキストはタグの内側なのでtextConentを使ったというわけですね。

////

19行目はにtimeoutIdにsetTimeoutが代入されている構造になっていると思うのですが、  
なぜsetTimeoutが実行されるのでしょうか？

少々ご認識が異なっているようです。  
正しくは、setTimeoutの実行結果がtimeoutIdに代入されるが正となります。

////

# timeoutIdはreturnする必要はないのですか？

eturnとは、関数に値を返却するという意味だと思っています。  
また値を返却することで、関数外でその値が利用できると認識してます。  
  
今回はfunction countUp() 関数について質問させていただいております。  
function countUp()の中で使っているtimeoutIdは、値を countUp() に返却しないと  
countUp() の中でしか使えない認識でおりました。  
  
しかし、関数の外にあるclearTimeout(timeoutId);でtimeoutIdの結果を利用しているので  
疑問に思いました。  
  
初回の質問で外部の 変数 timeoutId にtimeoutIdを保存しているので、  
関数の処理が終わっても値が保持されるため，returnで返さなくても機能するとご教示いただきました。  
  
質問①  
なぜtimeoutId を外部に宣言する必要があるのでしょうか。  
中に書くと煩雑になるからでしょうか。  
  
質問②  
①が正しい場合、もし中で変数 timeoutIdを宣言した場合は、関数の処理が終わっても値が保持されないので  
少し処理が煩雑になるが、return timeoutiid;は必要になる認識に齟齬はないでしょうか。  
  
お手数おかけいたしますが、よろしくお願いします。

 先生

関数が外部とデータをやり取りする方法は二つあります．

一つは「引数と返り値」を使う方法で，もう一つは「関数の外側の変数」にアクセスする方法です．

引数と返り値を使う場合は関数の呼び出し時に毎回明示的にデータを指定します．

let 結果を受け取る変数 = 関数(関数に渡す変数);

それとは別に，関数は関数外部の変数にアクセスすることができるため，関数外部の変数を読み取ったり書き換えたりすることでデータのやり取りができます．

returnとは、関数に値を返却するという意味だと思っています。  
また値を返却することで、関数外でその値が利用できると認識してます。

return を使うと，関数を呼び出す際に let retval = func(); のようにして結果を受け取ることができます．  
return がなくても関数は関数の外部の変数とデータのやり取りができます．(そうなの！？初耳！)

質問①  
なぜtimeoutId を外部に宣言する必要があるのでしょうか。  
中に書くと煩雑になるからでしょうか。

関数の内部で宣言された変数は関数の実行終了とともに破棄され，関数の外部からアクセスすることはできません．  
そのため，関数の実行終了後にも利用したい変数は必ず外部で宣言する必要があります．

質問②  
①が正しい場合、もし中で変数 timeoutIdを宣言した場合は、関数の処理が終わっても値が保持されないので  
少し処理が煩雑になるが、return timeoutiid;は必要になる認識に齟齬はないでしょうか。

そうですね．内部で宣言した場合は関数の終了とともに変数が破棄されてしまうので，返り値を使ってデータのやり取りをする必要があります．

/////

const timer = document.getElementById('timer');

としたあとに、

timer.textContent = `${m}:${s}.${ms}`;

としてもエラーになりません。

ここは間違えやすいポイントですが、constに関しては**再代入をするとエラーになります。**

したがって、例えば、

const timer = document.getElementById('timer');

としたあとに、

timer = 'hello';

とするとエラーになりますが（timerに再代入している）、

timer.textContent = 'hello';

としてもtimer自体に再代入しているわけではないのでエラーにはならない、ということですね。

////

タイマーを再開できるようにする。

タイマーをストップして再開するとゼロからのカウントとなってしまうのでストップで止まった時間から再開できるようにする。

Stopをクリックした時にclearTimeoutでタイマーを停止しつつ、タイマーが走っていた時間を保持しておき、またスタートがクリックされた際にそれを足し合わせれば良い。

Timerが走っていた時間をelapsedTimeとしてやる。(初期値は0)

Stopが押されたときにelapsedTimeを計算していけば良い。

Countupの方で経過時間も含めて表示させれば良い。

Stopを2回押してからstartを押すとタイマーが少し戻ってしまっている。これは

elapsedTimeが直近でタイマーが走っていた時間しか保持していなかったためなので、こちらをタイマーが走っていた時間全てを保持するようにする。

ストップしてからリセットして、再度スタートさせると前回ストップしたところから再開してしまう。Resetとした際にはelapsedtimeもリセットするようにする。

'use strict'

{

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

let startTime;

let timeoutId;

let elapsedTime = 0;

function countUp() {

// console.log(Date.now() - startTime);

const d = new Date(Date.now() - startTime + elapsedTime);

const m = String(d.getMinutes()).padStart(2, '0');

const s = String(d.getSeconds()).padStart(2, '0');

const ms = String(d.getMilliseconds()).padStart(3,'0');

timer.textContent = `${m}:${s}.${ms}`;

timeoutId = setTimeout(() => {

countUp();

}, 10);

}

start.addEventListener('click', () => {

startTime = Date.now();

countUp();

});

stop.addEventListener('click', () => {

clearTimeout(timeoutId);

elapsedTime += Date.now() - startTime;

});

reset.addEventListener('click', () => {

timer.textContent = '00:00.000';

elapsedTime = 0;

// startTime = Date.now();

});

}

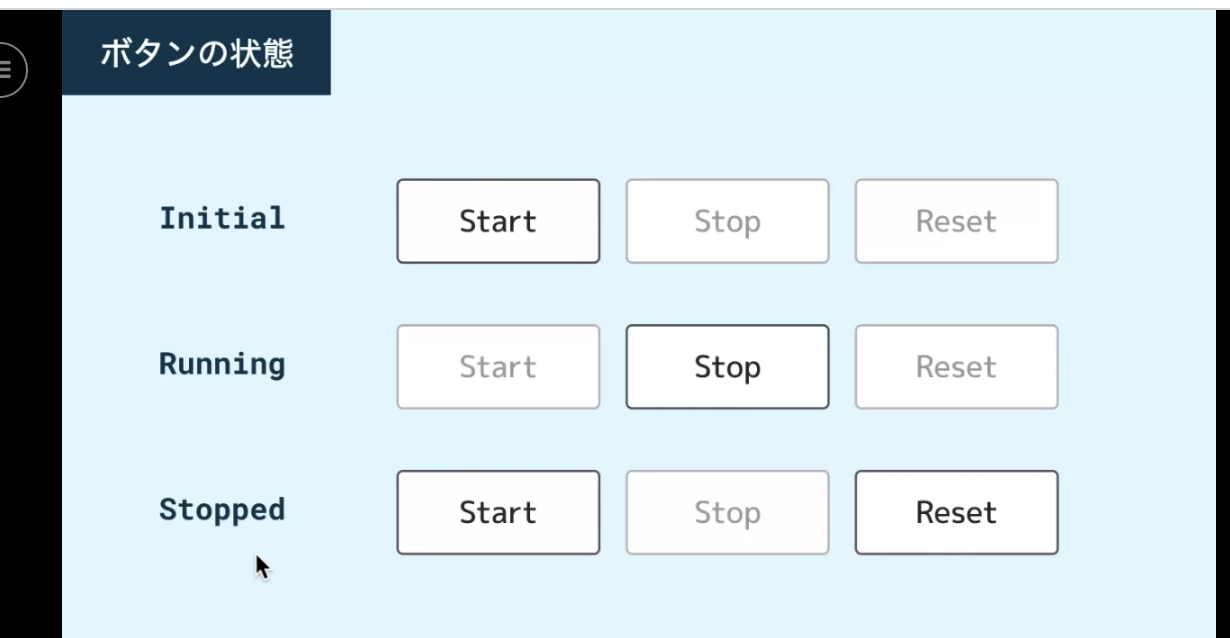
タイマーの問題点を把握する。ボタン要素のdisabled属性

ボタンを無効化していないことによる問題の把握。ボタンの状態。Button要素のdisabled属性。

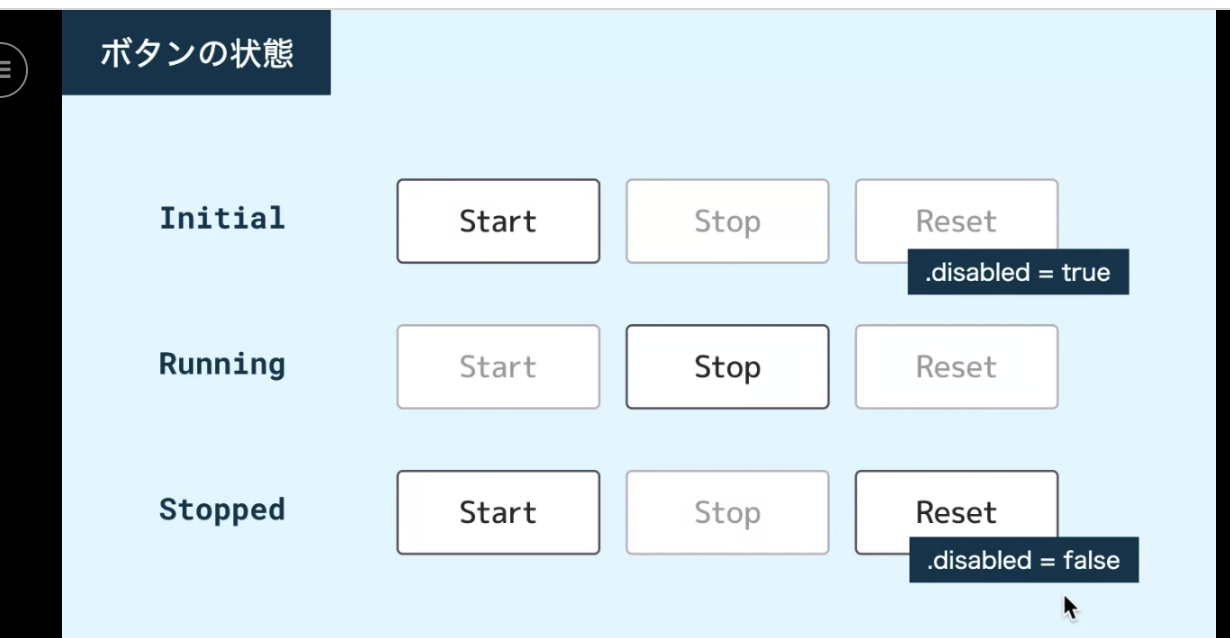
Startボタンを連打したときにstopボタンを1回押しても止まらない。Startボタンを押すたびに何度もストップウォッチが起動してしまうことから。

Startを押した後にstopを連打した場合にはelapsedtimeが新しい値で更新されてしまうので再開した時に少しおかしな値から始まってしまう。

これらはボタンが押せるべきでない時に押せてしまうことが原因。スタート前はスタートボタンだけ押せるようにし、それぞれのボタンを押したときに押すべきでないボタンは適宜無効化してあげれば良い。



ボタンのdisabledプロパティを用いて実装する。



ボタンの有効、無効を切り替える

function setButtonStateInitial(){

start.disabled = false;

stop.disabled = true;

reset.disabled = true;

}

function setButtonStateRunning(){

start.disabled = true;

stop.disabled = false;

reset.disabled = true;

}

function setButtonStateStopped(){

start.disabled = false;

stop.disabled = true;

reset.disabled = false;

}

setButtonStateInitial();

start.addEventListener('click', () => {

setButtonStateRunning();

startTime = Date.now();

countUp();

});

stop.addEventListener('click', () => {

setButtonStateStopped();

clearTimeout(timeoutId);

elapsedTime += Date.now() - startTime;

});

reset.addEventListener('click', () => {

setButtonStateInitial();

timer.textContent = '00:00.000';

elapsedTime = 0;

全体のスタイルを整える

Bodyのスタイリング、コンテナのスタイリング

<body>

<div class="container">

<div id="timer">00:00.000</div>

<button id="start">Start</button>

<button id="stop">Stop</button>

<button id="reset">Reset</button>

</div>

<script src ="js/main.js"></script>

</body>

Font-familyはタイマーの表示が崩れないように等幅フォントCourier Newを使う。総称フォントファミリーにも等幅フォントを指定する？monospace?

CSS

body {

font-family: 'Courier New', monospace;

font-size: 14px;

background: #eee;

}

.container {

margin: 20px auto;

width: 270px;

background: #fff;

padding: 15px;

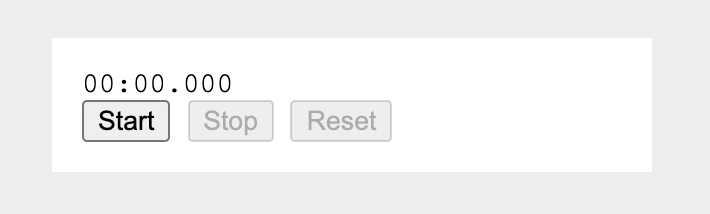
text-align: center;

}

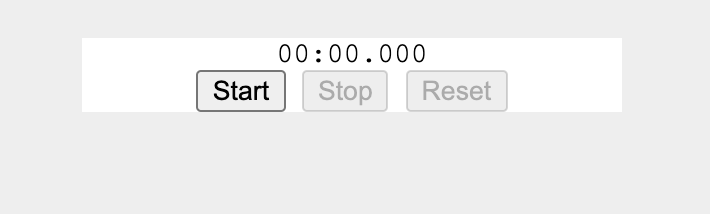
幅を指定しつつ中央揃えとしたいためmarginをセット。



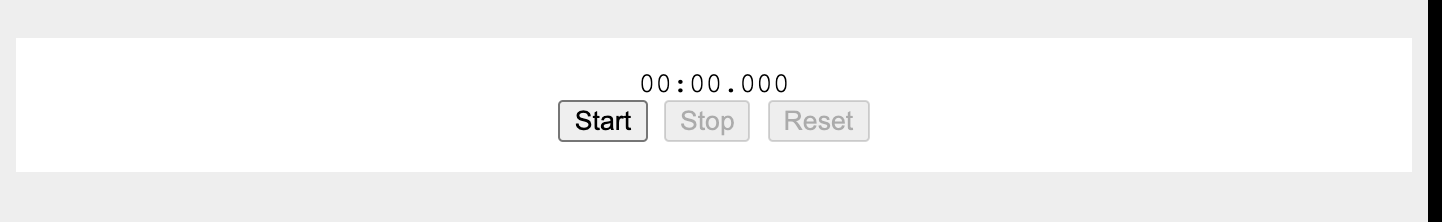
上記プログラム後



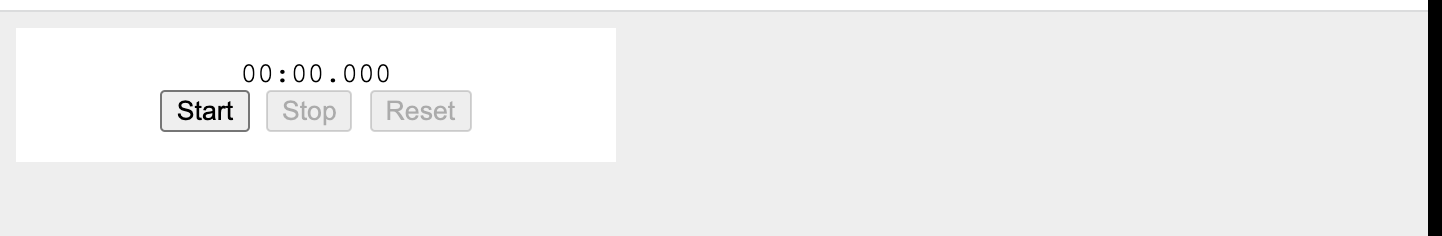
Text-alignがない場合



Paddingがない場合



Widthがない場合



Margin がない場合（autoで左右中央揃えに）

タイマーのスタイルを整える

Button要素をdiv要素に切り替える（div要素とbutton要素の違い）



Heightだけをセットした場合

#timer {

background: #ddd;

height: 120px;

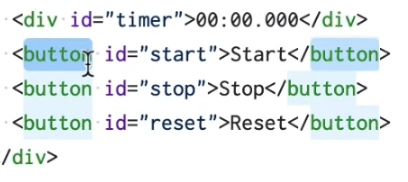
line-height: 120px;

font-size: 40px;

margin-bottom: 15px;

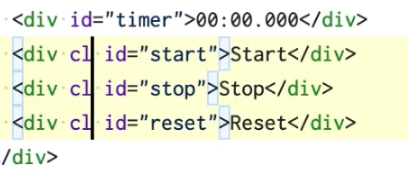
}

ボタンタグはブラウザーごとに独自のスタイルが設定されておりスタイル調整が少し難しいので、divタグで表現する。 ボタンっぽいスタイルをつけるためにクラスをつける。



Buttonあたりにカーソルを置いてcommand+Dで一気に選択が可能(複数回おすとひとつづつセレクトする。)

Command + option+↓で下方向にカーソルを追加でき一気に入力が可能



HTML

<body>

<div class="container">

<div id="timer">00:00.000</div>

<div class="btn" id="start">Start</div>

<div class="btn" id="stop">Stop</div>

<div class="btn" id="reset">Reset</div>

</div>

<script src ="js/main.js"></script>

</body>



中央揃えにしたいときには、line-heightプロパティがあればheightプロパティは必要ないのでしょうか？

中央揃えにする場合は line-height と height が同じ値になる必要があります．

しかし， height が指定されていない場合はデフォルトで height: auto; となっているので自動的に height が line-height と同じ値になるため， height を指定しない場合もうまく機能しています．

ただ，確実に同じ値にしておきたい場合は明示的に指定したほうが良いでしょう．

///

ボタンのスタイルを整える

ボタンを横に並べるためにflexboxを使う。そのためにボタンに対して親要素が必要となるのでdivで囲む。（classはcontrolsという名前にし、後でflexboxの設定をする）

間にスペースを持たせたいのでjustify-content : space-betweenとする。

ボタン連打するとテキストが選択されてしまいカッコ悪いのでcssのbtn classでuser-select: none;という設定を追加する。

Html

<body>

<div class="container">

<div id="timer">00:00.000</div>

<div class="controls">

<div class="btn" id="start">Start</div>

<div class="btn" id="stop">Stop</div>

<div class="btn" id="reset">Reset</div>

</div>

</div>

<script src ="js/main.js"></script>

</body>

Css

.btn {

width: 80px;

height: 45px;

line-height: 45px;

background: #ddd;

font-weight: bold;

cursor: pointer;

user-select: none;

}

.controls {

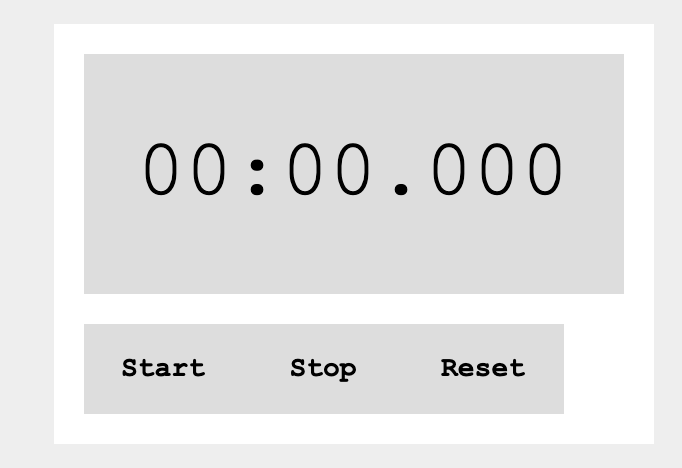
display: flex;

justify-content: space-between;

}



Justify-content：space betweenがない場合には下のようになる



# height と line-height の違いはなんですか？

今回の表示について言えば  
height を書かなくても同じ結果になりますね。

ただ両者は意味が違います。

* height は #timer 自体の高さ
* line-height は #timer に入るテキスト等の 1 行あたりの高さ

今回は #timer の中に経過時間を示すテキストしかありませんから  
height がなくても同じ結果になりますが、  
もし将来的にもっと複雑な配置にするようになった場合  
想定外の崩れ方をすることがあるかもしれません。

ただ、両者の意味の違いを把握した上で  
意図的に height を書かない、という選択をする可能性は考えられます。

「わかったうえで、あえてやる」のであれば問題ないと思います。

////

ストップウォッチを完成させる。

divタグを使ったボタンの有効・無効の切り替えを実装していきます。

* classList.add()
* classList.remove()
* classList.contains()

ボタンタグ（ボタン要素）ならdisabledプロパティを適宜使用していたが、**div要素を使う場合にはdisabledプロパティが使えない**

**代わりにクラスをつけたり外したりすることでスタイルをつけながら制御していく。**

ボタンが押せないように見えるスタイルはinactiveというクラスで作っていく

Javascriptの方でdisabled がtrueとなっている箇所にこのクラスをつける。

それぞれの要素に対してclassList.add(‘inactive’)と書き換える。

逆にdisabled = false となっている箇所は inactive クラスを外したいので Command + D でまとめて選択してあげて classList.remove('inactive') としてあげれば OK かと思います。

以上まででは見た目はOKだが操作を無効化するところまではできていない。（クリックを連続等したら変な挙動となる）

ボタンにinactiveクラスがついた場合には、それぞれの処理をしないようにすることで操作を無効化する。

CSS

.inactive {

opacity: 0.6;

}

JavaScript

function setButtonStateInitial(){

start.classList.remove('inactive');

stop.classList.add('inactive');

reset.classList.add('inactive');

}

function setButtonStateRunning(){

start.classList.add('inactive');

stop.classList.remove('inactive');

reset.classList.add('inactive');

}

function setButtonStateStopped(){

start.classList.remove('inactive');

stop.classList.add('inactive');

reset.classList.remove('inactive');

}

setButtonStateInitial();

start.addEventListener('click', () => {

if(start.classList.contains('inactive')===true) {

return;

}

setButtonStateRunning();

startTime = Date.now();

countUp();

});

stop.addEventListener('click', () => {

if(stop.classList.contains('inactive')===true) {

return;

}

setButtonStateStopped();

clearTimeout(timeoutId);

elapsedTime += Date.now() - startTime;

});

reset.addEventListener('click', () => {

if(reset.classList.contains('inactive')===true) {

return;

}

setButtonStateInitial();

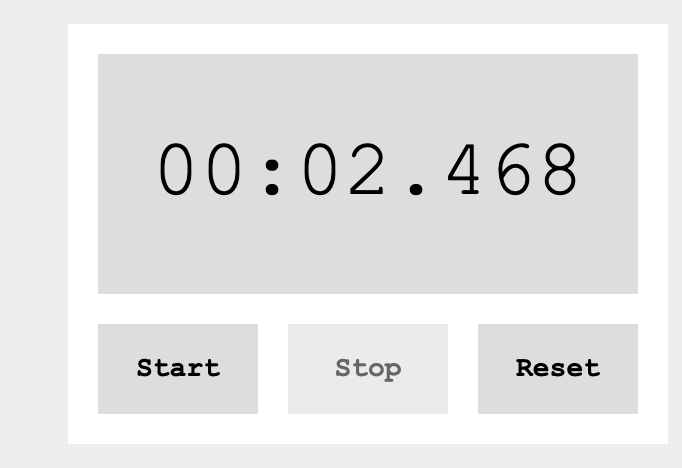
timer.textContent = '00:00.000';

elapsedTime = 0;

// startTime = Date.now();

});

完成



全コード

HTML

<!DOCTYPE html>

<html lang="ja">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Stopwatch</title>

<link rel="stylesheet" href="css/styles.css">

</head>

<body>

<div class="container">

<div id="timer">00:00.000</div>

<div class="controls">

<div class="btn" id="start">Start</div>

<div class="btn" id="stop">Stop</div>

<div class="btn" id="reset">Reset</div>

</div>

</div>

<script src ="js/main.js"></script>

</body>

</html>

JavaScript

'use strict'

{

const timer = document.getElementById('timer');

const start = document.getElementById('start');

const stop = document.getElementById('stop');

const reset = document.getElementById('reset');

let startTime;

let timeoutId;

let elapsedTime = 0;

function countUp() {

// console.log(Date.now() - startTime);

const d = new Date(Date.now() - startTime + elapsedTime);

const m = String(d.getMinutes()).padStart(2, '0');

const s = String(d.getSeconds()).padStart(2, '0');

const ms = String(d.getMilliseconds()).padStart(3,'0');

timer.textContent = `${m}:${s}.${ms}`;

timeoutId = setTimeout(() => {

countUp();

}, 10);

}

function setButtonStateInitial(){

start.classList.remove('inactive');

stop.classList.add('inactive');

reset.classList.add('inactive');

}

function setButtonStateRunning(){

start.classList.add('inactive');

stop.classList.remove('inactive');

reset.classList.add('inactive');

}

function setButtonStateStopped(){

start.classList.remove('inactive');

stop.classList.add('inactive');

reset.classList.remove('inactive');

}

setButtonStateInitial();

start.addEventListener('click', () => {

if(start.classList.contains('inactive')===true) {

return;

}

setButtonStateRunning();

startTime = Date.now();

countUp();

});

stop.addEventListener('click', () => {

if(stop.classList.contains('inactive')===true) {

return;

}

setButtonStateStopped();

clearTimeout(timeoutId);

elapsedTime += Date.now() - startTime;

});

reset.addEventListener('click', () => {

if(reset.classList.contains('inactive')===true) {

return;

}

setButtonStateInitial();

timer.textContent = '00:00.000';

elapsedTime = 0;

// startTime = Date.now();

});

}

CSS

body {

font-family: 'Courier New', monospace;

font-size: 14px;

background: #eee;

}

.container {

margin: 20px auto;

width: 270px;

background: #fff;

padding: 15px;

text-align: center;

}

#timer {

background: #ddd;

height: 120px;

line-height: 120px;

font-size: 40px;

margin-bottom: 15px;

}

.btn {

width: 80px;

height: 45px;

line-height: 45px;

background: #ddd;

font-weight: bold;

cursor: pointer;

user-select: none;

}

.controls {

display: flex;

justify-content: space-between;

}

.inactive {

opacity: 0.6;

}

div 要素を使う場合には disabled プロパティを使うことが出来ない」との内容でしたが、基本的にはどのような時に使えるのでしょうか？

 先生

disabledプロパティは基本的にユーザからのインプットを抑制するために利用します。例えば

* textareaに入力をさせない
* selectを選択させない
* ラジオボックスを選択させない

などといったことができるようになります。