タイピングゲームを作る

font-size プロパティに em が指定された場合はそうですね。親要素に対する相対的な値となります。

一方で、今回は letter-spacing プロパティに em が使われています。font-size 以外 に em が使用された際には、その要素の font-size に対する相対的な値となる仕様です。

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">

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

<title>Typing Game</title>

</head>

<body>

<p id="target">word</p>

<p id="result">Finished 10 sec</p>

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

</body>

</html>

Css

body {

padding-top: 40px;

font-family: 'Courier New', monospace;

text-align: center;

}

#target {

font-size: 48px;

/\* letter-spacing: .1em; \*/

}

タイプしたキーを表示させる

JS

"use strict";

{

document.addEventListener('keydown', e => {

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

target.textContent = e.key;

});

}

addEventListener()の第二引数は関数を渡しますね。  
実際にキーボードの何かkeyを押したときにkeydownイベントが発生し、渡した関数が実行されるのですが、この関数が実行されるときに引数としてeventオブジェクトを受け取ります。  
（このeventを慣習的にeとすることが多いです）

慣れないと少し難しいですが、addEventListener()の第二引数に渡す関数は引数にeventをとる、と覚えてしまうと良いかと思います。

**タイピングの正誤を判断する、正解した文字の表示を\_に変える**

(repeat()、substring())

文字列に対して配列のようにした赤字のように一文字一文字アクセスできることを知らなかった！

Locの個数分\_アンダーバーを繋げてあげれば良い。Repeat()というメソッドがあるのでそちらを活用。（\_をlocの個数だけ繋げた文字列を作ってくれる）

JSにはsubstringメソッドがあり、word.substring(loc)とすることでloc以降の文字列を取り出すことが可能。

{

const word = 'red';

let loc = 0;

//locationの意味

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

target.textContent = word;

document.addEventListener('keydown', e => {

if(e.key === word[loc]){

loc++;

console.log(word[2]);

// 1:\_ed

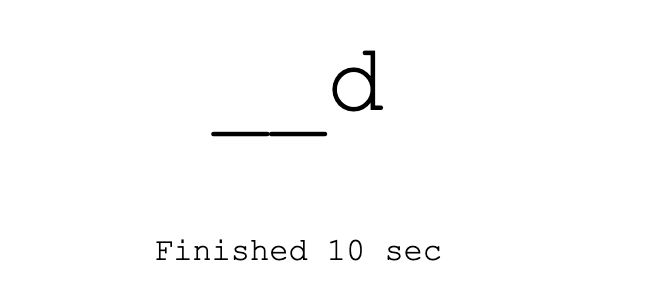
// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

}

});



word[loc]でwordのloc番目の文字列を指定しているという認識なのですが、  
wordを変数に指定して

word[loc] = '\_';

で直接wordのloc番目の文字列を\_に置き換えることはできないのでしょうか？

実際やってみたら、できませんでした。

 先生

こんにちは。ご質問ありがとうございました。

word[loc] = '\_';  
で直接wordのloc番目の文字列を\_に置き換えることはできないのでしょうか？

そうですね、

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

にも

ブラケット記法を使用した文字アクセスでは、これらのプロパティに値を設定したり削除したりすることはできません。関連したプロパティを書き込んだり設定したりすることもできません。

とありますね。こういう仕様になっているとご理解ください。

早期リターンを使う（早期リターンというテクニックを使ってコードを読みやすくしていく）

字下げが深くなっていて気になる。

不正解だった場合はそれ以降の処理をしなくて良いためこの時点でリターンとしてやる。その上でこの条件に該当しない場合には打ったキーは正解ということなので条件分岐を削ることができる。その後の処理は字下げをせずに記載することができる。

このようにメインとなる処理以外のケースを早めに除外してやり、メインとなる処理をわかりやすくすることを早期リターンという。もしくはアーリーリターンという。テクニックとして覚えておく。

{

const word = 'red';

let loc = 0;

//locationの意味

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

target.textContent = word;

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

});

}

////

早期リターンは、エラー処理などの、メインではない処理を最初にはじくために使われます。したがって、「この条件だったらA、そうでなかったらB」という処理において、AもBも同等の重要度がある場合は素直にif ... else ...を使ってあげると良いでしょう。

////

if (e.key !== word[loc]) {

return;

}

ifの条件（押したキーがwordのloc番目と同じではない）は理解できるのですが、その条件の時にreturnによってどのような処理をしているのかがよく分かりません。

 先生

main.jsの10行目から21行目のe => { }はアロー関数になっておりまして、関数のブロックの中でreturnを実行するとその時点で関数が終了します。つまり、e.key !== word[loc]の条件が正しい時に、returnが実行され、以下14行目以降の関数の処理は実行されません。

/////

今回は字下げが深く読みづらいために早期リターンを使ったのでしょうか。

if (e.key === word[loc]) {

loc++;

target.textContent = '\_'.repeat(loc) + word.substring(loc);

}

↑この表現だけでも結果は変わらないので、わざわざ早期リターンを使う必要性が理解できません。

このようにメインとなる処理以外のケースを早めに除外してあげて、メインとなる処理をわかりやすくする

とあるので、より複雑な条件分岐の際に使うとメリットは大きいのかと考えます。  
今回は簡単な条件のため、単に早期リターンの利点が分かりにくいということなのでしょうか。

 先生

↑この表現だけでも結果は変わらないので、わざわざ早期リターンを使う必要性が理解できません。

そうですね、今回はコードが短いのでどちらでもOKだと思います。

ただ、今後ゲームを改造するようなことを考えると、正常だった場合の処理が長くなる可能性もあります（正解部分を赤字にしたり、正解の音を鳴らしたり....）。そうしたことも考えて早期リターンを使っている、とご理解ください。

複数の単語に対応させる（複数の単語を定義、ランダムな要素の選択）

Wordsの中からランダム番目を選んでMath.floor()とMath.random()を使い0,1,2

の数値をランダムに出す。Words lengthが3なので、3\*(0~1)の値をかけたものなのでこれらの小数点以下をmath.floorで削ると0〜2の整数値が選ばれる

{

const words = [

'red',

'blue',

'pink',

];

let word;

let loc = 0;

//locationの意味

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

word = words[Math.floor(Math.random() \* words.length)];

target.textContent = word;

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

});

}

次の単語をセットする(setWord)

function setWord(){

word = words[Math.floor(Math.random() \* words.length)];

target.textContent = word;

loc = 0;

}

const words = [

'red',

'blue',

'pink',

];

let word;

let loc = 0;

//locationの意味

// const target = document.getElementById('target');

// word = words[Math.floor(Math.random() \* words.length)];

// target.textContent = word;

setWord();

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

if (loc === word.length) {

setWord();

}

});

最後まで打ち終わるとloc＝3でwordの数と同じであり、この後単語をセットしたい。とりあえずsetWordとおき、先頭の方で関数宣言しておく。新しい単語をセットしたらlocをリセットするためにloc=0;とする。

**単語の重複を防ぐ（splice()を使って単語が重複されないようにしたのち、ゲーム終了時の表示を作っていきます。）**

words.splice(Math.floor(Math.random() \* words.length), 1)で、「wordsの中からランダムに１要素を１つの配列として返している」

spliceの第一引数は配列のindex位置を表し、これをランダムに選択、第二引数でそのインデックス位置から取り除く要素数を指定する。取り除かないなら0。第三引数があった場合には代入する値である。Spliceの返り値は取り除かれた要素を含む配列である。Wordに配列を代入するのでなく要素を代入するので[0]としている。詳細はしたに説明あり。

3つ目の単語を打ち終えると単語がなくなって真っ白となってしまうため、条件分岐でwords.lengthが0となったら終了するように記載する。

function setWord(){

// word = words[Math.floor(Math.random() \* words.length)];

word = words.splice(Math.floor(Math.random() \* words.length), 1)[0];

target.textContent = word;

loc = 0;

}

const words = [

'red',

'blue',

'pink',

];

let word;

let loc = 0;

//locationの意味

// const target = document.getElementById('target');

// word = words[Math.floor(Math.random() \* words.length)];

// target.textContent = word;

setWord();

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

if (loc === word.length) {

if (words.length===0) {

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

result.textContent = 'Finished';

return;

}

setWord();

}

});

////

こちらの「返値」の説明にありますように、splice は取り除かれた要素を含む配列を返します。

一度に実施しているので分かりづらいのですが、分解すると次のような処理となります。

// この時点で words から1つ要素が取り除かれている。

// その要素を含む配列が tmp に格納される

const tmp = words.splice(Math.floor(Math.random()\*words.length),1);

// その要素を word に格納

word = tmp[0];

 ユーザー

すみません、やはりよくわかりませんでした。  
例えば、  
const words=[a,b,c];  
const word=words.splice(0,1)[0];  
とすると  
word='b'  
となるということでしょうか？  
  
この場合wordsからaが削除されているため、またsetWordが繰り返されるたび要素がひとつずつ減らされていってしまうと思うのですが...  
  
お手数おかけしますが、もう一度解説をよろしくお願いします。

 先生

const words = ['a', 'b', 'c'];

のとき、 words.splice(0, 1) で取り除かれるのは 'a' なので返り値は ['a'] となります。

その ['a'] の 0 番目を word に格納しますので word には 'a' が入りますね。

つまり1回目の、

const word = words.splice(0, 1)[0];

は、

const word = ['a'][0];

このように置き換えられるイメージです。

////

returnは{}で囲われたブロックの中だけの処理の実行を止めるのではなく、returnした箇所で関数を終了させるので、return以降のすべての処理は実行されません。

////

**ゲームをスタートさせる。（クリアタイムを表示させる）**

**toFixed()**

クリックした際に単語をセットし、時間も取得Date.now()。

最後の単語を打ち終わった時に経過時間を計算。ミリ秒で表示されるので1000で割って秒数にする。小数点第二まで表示したいのでtoFixed(2)を使用する。

"use strict";

{

function setWord(){

// word = words[Math.floor(Math.random() \* words.length)];

word = words.splice(Math.floor(Math.random() \* words.length), 1)[0];

target.textContent = word;

loc = 0;

}

const words = [

'red',

'blue',

'pink',

];

let word;

let loc = 0;

let startTime;

//locationの意味

// const target = document.getElementById('target');

// word = words[Math.floor(Math.random() \* words.length)];

// target.textContent = word;

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

startTime = Date.now();

setWord();

});

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

if (loc === word.length) {

if (words.length===0) {

const elapsedTime = ((Date.now() - startTime) / 1000).toFixed(2);

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

result.textContent = `Finished! ${elapsedTime} seconds!`;

return;

}

setWord();

}

});

}

////

tartTimeは定数にしてもいいのでは？

2021年1月16日

 ユーザー

動画では変数StartTimeを宣言したのちクリックのイベントリスナーでstartTimeに代入していましたが、自分はstartTimeの値が変わることはないと考えたので、クリックのイベントリスナーで定数としてstartTimeを宣言し、Date.now()で初期化しました。しかし、自分の方法だと最後のテキストが何も出力されません。どうしてでしょうか。

 先生

自分はstartTimeの値が変わることはないと考えたので

そうですね． startTime は startTime = Date.now(); で初期化されて以降変更されないのでconstにできると考えるのは自然かと思います．  
しかしながら結論としては今回はconstを使うことができません．

constは宣言と同時に初期化をする必要があるので，以下のように宣言をする形になるかと思います．

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

const startTime = Date.now();

setWord();

});

しかしながら，上記の位置で宣言してしまうと変数の有効範囲がこの () => { ... } の内部のみになってしまいます．  
そのため

if (words.length === 0) {

const elapsedTime = ((Date.now() - startTime) / 1000).toFixed(2);

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

result.textContent = `Finished! ${elapsedTime} seconds!`;

return;

}

の部分で startTime を参照することができずにエラーとなってしまいます．

そこで startTime の有効範囲を広げるためにお手本コードでは () => { ... } の外側である17行目で宣言しています．  
しかしこの位置ではまだ初期化できないため，letで宣言しています．

////

 先生

new Date() は Date オブジェクトを生成するのに対して  
Date.now() はこのような数値を返します。

Date.now() メソッドは、UTC (協定世界時) での 1970 年 1 月 1 日 0 時 0 分 0 秒 から現在までの経過時間をミリ秒単位で返します。  
<https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Global_Objects/Date/now>

今回は日時操作がしたいのではなく  
単に経過の秒数を取得したいだけなので  
Date オブジェクトを生成するのではなく  
Date.now() で得られた数値の差分を計算しています。

Date オブジェクトは日時を操作するだけでなく  
特定の日時を保持したり画面に表示したりするときにも使われます。

こちらのレッスンでもそのように使っていますね。  
[JavaScriptでカレンダーを作ろう (全18回)](https://dotinstall.com/lessons/calendar_js)

基本的に、日時を扱うときは Date オブジェクトを使うと思ってください。

Date.now() から数値を取得するのは  
今回のように秒数の差分だけが取りたいときなど特定の場面です。

**完成：ゲームの不具合を修正する**

**（ゲームの途中でもクリックできてしまう）**

変数isPlaying をfalseとして開始し、クリックしたらtrueにかえ、trueの場合には以後クリックイベントの中身を実行させないよう条件分岐で記載(return)する。

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">

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

<title>Typing Game</title>

</head>

<body>

<p id="target">Click to Start</p>

<p id="result"></p>

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

</body>

</html>

JavaScript

"use strict";

{

function setWord(){

// word = words[Math.floor(Math.random() \* words.length)];

word = words.splice(Math.floor(Math.random() \* words.length), 1)[0];

target.textContent = word;

loc = 0;

}

const words = [

'red',

'blue',

'pink',

];

let word;

let loc = 0;

let startTime;

let isPlaying = false;

//locationの意味

// const target = document.getElementById('target');

// word = words[Math.floor(Math.random() \* words.length)];

// target.textContent = word;

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

if (isPlaying === true) {

return;

}

isPlaying = true;

startTime = Date.now();

setWord();

});

document.addEventListener('keydown', e => {

if(e.key !== word[loc]){

return;

}

loc++;

// 1:\_ed

// 2:\_\_d

// 3:\_\_\_

target.textContent = '\_'.repeat(loc) + word.substring(loc);

if (loc === word.length) {

if (words.length===0) {

const elapsedTime = ((Date.now() - startTime) / 1000).toFixed(2);

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

result.textContent = `Finished! ${elapsedTime} seconds!`;

return;

}

setWord();

}

});

}

CSS

body {

padding-top: 40px;

font-family: 'Courier New', monospace;

text-align: center;

}

#target {

font-size: 48px;

/\* letter-spacing: .1em; \*/

}