付録 A

「練習問題」「この章の理解度チェック」解答

第１章の解答

この章の理解度チェック　p.017

[1] ECMAScriptとは

ECMAScriptは、JavaScriptのコアの仕様をまとめた言語仕様。

[2] ECMAScriptとバージョン

ES6（ES2015）

[3] JavaScript実行環境

ブラウザ環境、Node.js環境

[4] 実行環境ごとの機能の違い

ブラウザ環境：Web API、Node.js環境：CommonJS

[5] 実行環境の違いに伴う留意点

それぞれの環境でのみ準備されているAPI（たとえば、Web APIやCommonJS）の機能を使おうとするとエラーになること。

[6] ツールの役割

Visual Studio Codeは、コードの記述や編集を行います。Live Serverは、開発を行うPC上で仮想のサーバーを起動して、ブラウザからのリクエストを受け取れる状態にします。Chromeは、記述したコードの画面上での確認の他、開発ツールを使ったエラーの原因調査やコードの実行状況の確認ができます。

第２章の解答

この章の理解度チェック　p.037

[1] コード記述のルール

① 大文字小文字
② 半角文字
③ セミコロン（;）
④ `/* */`

[2] コンソールログの確認　配布サンプル `sec_end2.html`

```
263
```

[3] エラーの確認　配布サンプル `sec_end3.html`

`Val3`の頭文字が大文字のため。

解説 前の行で定義している`val3`は、頭文字が小文字のため、`Val3`とは別の変数とみなされています。`ReferenceError`は変数や関数が取得できなかった場合に発生するので、`Val3`が取得できずエラーになっています。

[4] エラーの原因

ブレイクポイントで止まった行はまだ実行されていない状況のため。

解説 `let val3 = 789 % 3;`の行でコードの実行が止まっている場合、［Step Over］などを使って、次の行に処理を進めないと、`val3`の値は確認できません。

[5] 開発ツールでの確認

`0`が確認できれば正解です。

解説 4行目が実行完了した状態で、コンソールに`val3`と入力してEnterキーを押せば、`val3`の値がコンソールに表示されます。

第3章の解答

練習問題 3.1 p.048

[1] `let`は値の再代入が可能ですが、`const`は値の再代入を行うことはできません。

[2] **配布サンプル** `const_error.html`

エラーは2行目（`fruit = "banana";`）で発生します。`const`を使っているので、値の再代入ができないため、エラーが発生します。

練習問題 3.2 p.052

[1] **配布サンプル** `camel_case.html`

次のように、2つ目以降の単語の先頭を大文字にします。

▶解答例

```
// キャメルケース
let productPrice = 1000;
let cartItem = "りんご";
let favoriteSportCategory = "球技";
```

練習問題 3.3 p.058

[1] **配布サンプル** `declare_string.html`

▶解答例

```
 // エスケープを使う場合
let str = 'This is Tom\'s house.';
console.log( str );

// エスケープを使わない場合
let str = "This is Tom's house.";
console.log( str );
```

練習問題　3.4　p.060

[1] 配布サンプル　print_calc.html

①
```
let sum = 5 + 6 - 1;
console.log( sum );
> 10
```

②
```
sum = sum * 2;        // 10 * 2
console.log( sum );
> 20
```

③
```
sum = sum % 3;        // 20 / 3の余り2
console.log( sum );
> 2
```

④
```
sum = sum ** 3;        //  2 * 2 * 2 = 8
console.log( sum );
> 8
```

解説　左辺と右辺に同じ変数sumが使われていますが、**代入演算子（=）は右辺の結果を左辺に代入する**ため、右辺の計算結果が新しいsumの値として設定されます。

練習問題　3.5　p.064

[1] シングルクォート（'）、ダブルクォート（"）、バッククォート（`）

[2] 16進数

[3] 正しい

解説　0.123という小数値がコンソールに表示されます。

[4] 正しくない

解説　数値とBigIntは、同じ文で使えません。

[5] true、false

[6] nullは、参照を保持していないことを表します。
undefinedは、変数が未定義であることを表します。

練習問題　3.6　p.072

[1] 配布サンプル declare_object.html（[1] ～ [3] 含む）

▶解答例

```
let obj = {
    prop: true,
    subObj: { val: 100 },
    greeting: function () {
        console.log("こんにちは。");
    },
};
```

[2] ▶解答例

```
console.log( obj["prop"] );
console.log( obj["subObj"]["val"] );
obj["greeting"]();
```

[3] ▶解答例

```
// ドット記法の場合
obj.subObj.val2 = 1000;

// ブラケット記法の場合
obj["subObj"]["val2"] = 1000;
```

練習問題　3.7　p.078

[1] 配布サンプル falsy_value.html

```
console.log( Boolean(0) );
> false
console.log( Boolean("") );
> false
console.log( Boolean(null) );
> false
console.log( Boolean(undefined) );
> false
```

練習問題　3.8　p.079

[1] 配布サンプル　implicit_conv.html

① 1　　解説 trueが1に変換されます。

② 1　　解説 falseが0、trueが1に変換されます。

③ false　解説 数値の0はfalseに変換されます。

④ true　解説 数値の1はtrueに変換されます。

⑤ true　解説 数値の0以外はtrueに変換されます。

この章の理解度チェック　p.080

[1] 変数

① アドレス

② let

③ const

④ var

[2] 文字列の操作　配布サンプル　sec_end2.html

▶解答例

```
const TAX_RATE = 1.1;
let productPrice = 1000;
let priceWithTax = productPrice  * TAX_RATE;

// 加算演算子の場合　※シングルクォートを使っても正解
console.log( "商品の金額は" + productPrice + "円ですので、税込金額は" + priceWithTax +
"円です。" );

// テンプレートリテラルの場合
console.log( `商品の金額は${productPrice}円ですので、税込金額は${priceWithTax}円です。` );
```

[3] 数値計算　配布サンプル　sec_end3.html

① 4 (12 ** 2 % 5)

② 4n (12n ** 2n % 5n)

③ 文字列を数値に変換しようとしたとき。

▶例

```
console.log( "string" - 1 );
> NaN
```

[4] nullとundefined

① 空（参照を保持していない状態）

② 開発者

③ 未定義

[5] オブジェクトの記法 配布サンプル sec_end5.html

▶解答例

```
// ドット記法の場合
const counter = { num: 1 };
counter.num = counter.num + 1;
window.alert( counter.num );

// ブラケット記法の場合
const counter = { num: 1 };
counter["num"] = counter["num"] + 1;
window.alert( counter["num"] );
```

[6] 型変換 配布サンプル sec_end6.html

① object 解説 nullは、objectと表示されます。

② 101 解説 trueは、数値の1に暗黙的に変換されます。

③ NaN 解説 Number("hello")は、数値に変換できないため、NaNが返されます。

④ 2 解説 Boolean("hello") => trueになり、trueは数値の1に暗黙的に変換されます。

第4章の解答

練習問題 4.1 p.087

[1] 配布サンプル group_ope.html

aは11、bは10

解説 グループ化演算子( )によってb = 10が最優先で処理されます。そのため、変数bの値は10になります。b = 10の代入演算子は10を結果として返すため、1 + 10が次に実行されます。そのため、変数aの実行結果は11になります。

練習問題 4.2 p.089

[1] 配布サンプル calc_ope.html

① 18

② 6

③ 0

④ 1 解説 コンソールログで1が出力されてから+1されます。

⑤ 18 解説 a – 1掛けるb + 1の結果になります。

練習問題　4.3　p.093

[1] 配布サンプル comp_ope.html

① false

② true

③ 3

解説 ③の(9 > 3) * 3;では、まずグループ化演算子によって9 > 3が最初に実行されます。この結果はtrueですが、trueは暗黙的な型変換により数値の1とみなされ、1 * 3になり、3が導き出されます。

練習問題　4.4　p.095

[1] 配布サンプル assing_ope.html

① 3Ø

② 15

③ 5

④ 2

⑤ 8（2**3 = 2^3）

練習問題　4.5　p.100

[1] 配布サンプル single_ope.html

① { age: "32", male: true }

② object

③ 32（数値型）

④ false

この章の理解度チェック　p.105

[1] 演算子とオペランド

① 加算

② オペランド

③ 単項

[2] 演算子の優先順位　配布サンプル end_sec2.html

▶解答例

```
console.log( (1Ø + 5) * 4 );
```

解説 グループ化演算子を使って、1Ø + 5の処理の優先順位を上げてあげましょう。

[3] インクリメント演算子とデクリメント演算子 **配布サンプル** end_sec3.html

インクリメント演算子（++）は、オペランドに指定した変数をインクリメント（1を加算）する働きを持つ演算子ですが、演算子をオペランドの前に置くか後ろに置くかで、式として返す値が異なるためです。x++とすると、変数xの値はインクリメントされますが、式として返す値はインクリメントする前の値です。一方、++yとすると、変数yの値はインクリメントされ、式として返す値もインクリメントされた後の値が返されます。

[4] 厳密な等価性と抽象的な等価性

① false

② true

③ 型（データ型）

[5] 代入演算子 **配布サンプル** end_sec5.html

① console.logによる出力結果は、**undefined**です。変数を宣言するときに値を代入しない場合には、undefinedがプログラムによって自動的に設定されるためです。

② console.logによる出力の結果は、10になります。これは、「a = 10という代入演算が変数aに値10を設定する」という働きを持つだけでなく、「a = 10という式で代入した値を返す」ためです。今回は10を代入しているため、代入演算式の返す値は10になります。

③ console.logによる出力の結果は、10になります。②のa = 10という代入演算の結果、変数aには10という値が設定されたためです。

[6] 自己代入演算子 **配布サンプル** end_sec6.html

① a += 3;

② a *= 4;

③ a /= 2;

④ a %= 5;

[7] 論理演算子 **配布サンプル** end_sec7.html

① undefined

② こんにちは

[8] その他の演算子 **配布サンプル** end_sec8.html

① バナナ

解説 ?の前の条件式（null）がfalsyな値のため、三項演算子の結果は:の右側の値（"バナナ"）になります。

② パイナップル

解説 ??は、Null合体演算子です。そのため、??の左オペランドの値がnullまたはundefinedの場合には、右オペランド（"パイナップル"）が??の結果になります。

③ バナナ

解説 { apple: "リンゴ" }?.fruitの部分は、オプショナルチェイニング演算子によって{ apple: "リンゴ" }オブジェクトからfruitプロパティを取得しようとしています。このとき、fruitプロパティは取得できず、undefinedが返ってくるため、その後に続くNull合体演算子（??）によって"バナナ"が取得されます。

第５章の解答

練習問題　5.1　p.116

[1]　配布サンプル　if_logical.html

▶解答例

```
const person = {
    name: "Bob",
    age: 28,
    gender: "male"
}

if (person.gender === "male" && person.age >= 25) {

    console.log( `${person.name}は25歳以上の男性です。` );

} else {

    console.log( `${person.name}は25歳以上の男性ではありません。` );

}
```

練習問題　5.2　p.117

[1]　配布サンプル　set_default_val.html

①　▶解答例

```
// 解答例1　if文を使った場合
let val = "";
if( !val ) {
    val = "Hello";
}
console.log( val );

// 解答例2　論理和を使った場合
let val = "";
val = val || "Hello";
```

② ▶解答例

```
// 解答例1  if文を使った場合
let val = null;
if( val === undefined || val === null ) {
    val = "Hello";
}

// 解答例2  Null合体演算子を使った場合
let val = null;
val = val ?? "Hello";
```

練習問題　5.3　p.120

［1］ 配布サンプル switch.html

▶解答例

```
let animal = "ゾウ";  // 変数の初期値はなんでもよい

switch( animal ) {

    case "ゾウ":
        console.log( "ゾウ" ) ;

    case "ウマ":
        console.log( "ウマ" ) ;

    case "ウサギ":
        console.log( "ウサギ" ) ;
        break;

    default:
        console.log( "何かわかりません。" ) ;

}
```

練習問題　5.4　p.127

［1］ 配布サンプル　exception_handling.html

```
try {

    let b = 10 + a;    // 宣言していない変数を使用しているため、ReferenceErrorが発生！
    console.log( b );

// 例外識別子には任意の名前を付けることができる
} catch( e ) {

    console.error( e );   // エラーをコンソールに出力

} finally {

    console.log( "後続の処理" );

}
// finallyブロックを設定しない場合は、ここに後続処理に記述してもよい
// console.log( "後続の処理" );
```

練習問題　5.5　p.129

［1］ 配布サンプル　throw_error.html

▶解答例

```
// greetingに値を設定しないと例外に飛ぶ
// greetingに "こんにちは" などの文字列を設定すると例外は発生しない
let greeting;

try {

    if( typeof greeting !== "string" ) {
        throw new Error( "不正なデータ型です。" );
    }

    console.log(`${ greeting }、いい天気ですね。`);

} catch( e ) {
    console.error( e );
}
```

練習問題　5.6　p.132

[1]　配布サンプル　while_by_2.html

▶解答例

```
let i = 0;
while( i <= 6) {
    console.log( i );
    i += 2;
}
```

練習問題　5.7　p.134

[1]　配布サンプル　for_by_3.html

▶解答例

```
for(let i = 0; i <= 9; i += 3) {
    console.log( i );
}
```

練習問題　5.8　p.136

[1]　配布サンプル　sum_array_elem.html

▶解答例

```
const arry = [ 10, 20, 23, 47 ];
let sum = 0;    // sumに値を加算していく

for( let i = 0; i < arry.length; i++ ) {
    const value = arry[ i ];
    sum += value;
}

console.log( sum );
> 100
```

練習問題　5.9　p.137

[1]　配布サンプル　sum_for_in.html

▶解答例

```
const obj = {
    prop1: 10,
    prop2: 20,
    skip: 20,
    prop3: 23,
    prop4: 47,
};

let sum = 0;    // sumに値を加算していく

for( const key in obj ) {
    if( key !== "skip" ) {
        sum += obj[ key ];
    }
}

console.log( sum );
> 100
```

練習問題　5.10　p.140

[1]　配布サンプル　length_not_enumerable.html

以下のように、enumerableの値を確認できていれば正解です。

▶解答例

```
const arry = [ ];    // 空の配列を定義
const propDesc = Reflect.getOwnPropertyDescriptor( arry , "length" );   // 記述子を取得
console.log( propDesc.enumerable );    // 列挙可能性を確認
> false
```

練習問題　5.11　p.142

[1]　配布サンプル　sum_for_of.html

▶解答例

```
const arry = [ 10, "文字列", 20, true, 23, 47 ];
let sum = 0;    // sumに値を加算していく

for( const value of arry ) {
    if( typeof value === "number" ) {
        sum += value;
    }
}

console.log( sum );
> 100
```

練習問題　5.12　p.145

[1]　配布サンプル　sum_object_elem.html

▶解答例

```
const obj = {
    prop1: 10,
    prop2: "文字列",
    prop3: 20,
    skip: 20,
    prop4: true,
    prop5: 23,
    prop6: 47,
};

let sum = 0;    // sumに値を加算していく

const entries = Object.entries( obj );
for( const entry of entries ) {
    const key = entry[ 0 ];
    const value = entry[ 1 ];
    if( typeof value === "number" && key !== "skip"  ) {
        sum += value;
    }
}

console.log( sum );
> 100
```

練習問題　5.13　p.146

[1]　配布サンプル　break_for_of.html

▶解答例

```
const breakTestArry = [ "ぬけない", "not break", "break", "この前で抜ける" ];
for( const value of breakTestArry ) {
    console.log( value );
    if( value === "break" ) {
        break;
    }
}
> ぬけない
> not break
> break
```

練習問題　5.14　p.147

[1]　配布サンプル　continue_for.html

▶解答例

```
for ( let i = 1; i < 5; i++ ) {    // 1からループを始める
    if ( i === 3 ) {    // 3回目
        continue;
    }
    console.log( `${ i }回目のループです。` );
}
```

この章の理解度チェック　p.149

[1]　for文　配布サンプル　sec_end1.html

▶解答例

```
for ( let i = 5; i < 10; i++ ) {
    console.log( i );
}
```

[2] If...elseチェーン　配布サンプル　sec_end2.html

この問題は「FizzBuzz問題」としてよく知られているものです。以下のように実装すると、条件どおりコンソールに文字が出力されます。

▶解答例

```
for (let i = 1; i <= 100; i++) {
    if (i % 3 === 0 && i % 5 === 0) {
        console.log("Fizz Buzz");
    } else if (i % 3 === 0) {
        console.log("Fizz");
    } else if (i % 5 === 0) {
        console.log("Buzz");
    } else {
        console.log(i);
    }
}
```

[3] for...in文と列挙可能性

① 列挙可能

② プロパティ記述子（またはディスクリプタ）

③ enumerable

[4] for...of文　配布サンプル　sec_end4.html

以下は、本章で学習したfor...of文やObject.entriesを使った解答例です。

▶解答例

```
const capitals = {
    日本: "東京",
    アメリカ: "ワシントン",
    イギリス: "ロンドン"
};

for (const [ country, capital ] of Object.entries( capitals )) {
    console.log( `${country}の首都は${capital}です。` );
}
> 日本の首都は東京です。
> アメリカの首都はワシントンです。
> イギリスの首都はロンドンです。
```

[5] 例外処理 配布サンプル sec_end5.html

コンソールへの出力結果は、次のようになります。

```
> tryブロックの処理を開始します。
> catchブロックの処理を開始します。
> catchした値：例外を投げました。
> catchブロックの処理を終了します。
> finallyブロックの処理を実行します。
> try/catch/finally文の後続のコードを実行します。
```

tryブロック内で例外を発生させる（投げる）とtryブロック内のそれ以降の処理は実行されないため、「tryブロックの処理を終了します。」というコンソールへの出力処理は実行されません。

throw文が実行されると、処理はcatchブロックに移ります。そのとき、例外識別子eにはthrowした値が渡されるため、eには"例外を投げました。"という文字列が格納されることになります。

catchブロックの処理が終了すると、finallyブロックの処理が実行されます。finallyブロックは、例外が発生せずにcatchブロックが実行されなかったときでも必ず実行されます。

finallyブロックが終了すると、try/catch/finally文の後続のコードが実行されます。

第6章の解答

練習問題 6.1 p.154

[1] 配布サンプル multiply.html

▶解答例

```
function multiply( val1, val2 ) {
    return val1 * val2;
}

console.log( multiply( 7, 9 ) );
> 63
console.log( multiply( -11, 9 ) );
> -99
```

練習問題　6.2　p.156

[1] 配布サンプル　args_behavior.html（[1] ～ [3] 含む）

▶解答例

```
function noArgumentFunc() {
    console.log( "引数がない関数です。" );
}

noArgumentFunc();
> 引数がない関数です。
```

[2] ▶解答例

```
function multiply( val1, val2 ) {
    console.log( val1 * val2 );
}

multiply( 2, 3 );
> 6
multiply( 15, 2, 10 );
> 30
```

解説 multiply(15, 2, 10);は、15 * 2 = 30となり、第3引数の10は無視されていることがわかります。

[3] 実引数の個数と仮引数の個数が一致しない場合にもエラーは発生せず、前方にある引数から順番に値を受け取り、受け取れなかった後方の引数には自動的にundefinedが設定されます。

```
twoArgumentFunc();
> arg1: undefined
> arg2: undefined

twoArgumentFunc( 10 );
> arg1: 10
> arg2: undefined

twoArgumentFunc( 10, 20 );
> arg1: 10
> arg2: 20
```

練習問題　6.3　p.158

[1]　配布サンプル　return_behavior.html（[1] ～ [2] 含む）

▶解答例

```
function hello() {
    return "Hello World";
}

console.log( hello() );    // 関数が実行された結果（戻り値）がコンソールに出力される
> Hello World

// 一度変数に戻り値を格納することもできる
const result = hello();
console.log( result );
> Hello World
```

[2]

```
function hello( personName ) {
    if( typeof personName !== "string" ) {
        console.log( "引数に文字列を渡してください。" );
        return;    // 関数の処理を終了
    }
    console.log( `こんにちは、${ personName }` );
}
hello( "太郎" );
> こんにちは、太郎
hello( 1 );
> 引数に文字列を渡してください。
```

解説　解答例ではreturn文を使っていますが、if...else文を使って記述してもかまいません。

[3]　fn1、fn2のどちらも、戻り値はundefinedになります。

解説　関数の戻り値とは、関数が実行された結果の値のことです。fn1の場合は、return文がないため、関数の実行結果はundefinedとなります。fn2の場合は、return;のようにreturn文で値が設定されていないため、この場合にも戻り値はundefinedになります。関数内でコンソールに出力するなどの操作は、あくまで関数の処理であり、戻り値とは関係ないため、注意してください。

練習問題　6.4　p.162

[1]　配布サンプル　function_declare.html（[1] ～ [2] 含む）

▶解答例　関数宣言の場合

```
console.log( `3 + 5 = ${sum(3, 5)}` );
> 3 + 5 = 8

function sum( val1, val2 ) {
    return val1 + val2;
}

console.log(`10 + 7 = ${sum(10, 7)}`);
> 10 + 7 = 17
```

[2]　▶解答例　関数式の場合

```
console.log(`3 + 5 = ${sum(3, 5)}`);    // エラーが発生！
> Uncaught ReferenceError: Cannot access 'sum' before initialization
                                    [意訳] 参照に関するエラー：sumは初期化前にアクセスできません。
// エラーが発生したため、これ以降は実行されない

const sum = function ( val1, val2 ) {
    return val1 + val2;
}

console.log(`10 + 7 = ${sum(10, 7)}`);
```

練習問題　6.5　p.164

[1]　配布サンプル　circle.html

▶解答例

```
function calcAreaOfCircle( radius, pi = 3 ) {
    return pi * radius ** 2;
}

console.log( calcAreaOfCircle( 10, 3.14 ) );
> 314
console.log( calcAreaOfCircle( 10 ) );
> 300
```

練習問題　6.6　p.168

[1] 配布サンプル　ref_obj.html

コンソールの出力結果は、次のとおりです。

```
> 3 ―――――――――――――――――――――――――――――――――――――――――――― ①
> 5 ―――――――――――――――――――――――――――――――――――――――――――― ②
> 3 ―――――――――――――――――――――――――――――――――――――――――――― ③
```

① console.logの出力は3です。

解説　関数fnの引数object1Argには、{ num: 3 }が格納されているメモリ空間のアドレスが渡されますが、関数fn内の処理でobject1Argは**別のオブジェクト**{ num: 5 }で上書きされています。そのため、関数外の変数obj1の{ num: 3 }に影響を及ぼすことはできません。

② console.logの出力は5です。

解説　関数fnの引数object2Argには、{ num: 3 }が格納されているメモリ空間のアドレスが渡されます。そして、関数fn内の処理では、object2Arg.num = 5;により、オブジェクトの中身が変更されています。この変更を行った時点では、obj2とobject2Argは同じオブジェクトを参照していることになるため、関数内の変更が関数外（obj2）にも影響します。

③ console.logの出力は3です。

解説　関数fnの引数numberArgには、数値の3（正確には3が格納されているメモリ空間のアドレス）が渡されます。これが関数fn内の処理でnumberArgは5（5が格納されているメモリ空間のアドレス）で上書きされています。このように、**プリミティブ値（数値や文字列）が引数として渡された場合には、関数内部の変更は実引数として渡した変数（num）には影響しません。**この挙動は、3.2.4項（p.046）で紹介した次のコードで説明できます。

```
let hello = "こんにちは";
let greeting = hello;
let greeting = "さようなら";
```

この場合には、変数greetingに変数helloの値を格納して、その後greetingの値を変更していますが、この操作によって変数helloの値が変わることはありません。この練習問題でも引数としてnumからnumberArgに値を渡していますが、その後、関数内でnumberArgの値を変更してもnumの値は変わりません。

練習問題　6.7　p.170

[1] 配布サンプル　fn_variable.html

実行結果は、次のとおりです。

```
> こんにちは、独習太郎
```

const obj = hello;によって、関数helloを変数objに代入しています。このコードは記述内容としては問題ないため、エラーは発生しません。そのため、関数はオブジェクト{ }や数値、文字列などのプリミティブ型の値と同様に、別の変数に代入できることがわかります。また、objの末尾に("独習太郎")のように丸括弧を付けることで、関数を実行できます。これらの事実から、関数はオブジェクトと同じく他の変数に代入したりできることに加えて、他の変数に代入しても関数として実行可能であることがわかります。

練習問題 6.8 p.173

[1] 配布サンプル delay_hello.html

▶解答例

```
const hello = function() {
    console.log( "こんにちは" );
}

setTimeout( hello, 5000 );
```

練習問題 6.9 p.174

[1] 配布サンプル callback_calc.html

▶解答例

```
function plus(a, b) { return a + b; }
function minus(a, b) { return a - b; }

function calc( val1, val2, callback ) {
    // callbackにplusやminusなどのコールバック関数が渡され、実行される
    const result = callback( val1, val2 );
    console.log( result );
    // 以下のように記述しても正解
    // console.log( callback( val1, val2 ) );
}

calc( 1, 2, plus );
> 3
calc( 10, 2, plus );
> 12
calc( 10, 2, minus );
> 8
```

練習問題　6.10　p.176

[1]　配布サンプル　anonymouse.html

①
```
const hello = function() { console.log( "こんにちは" ); }
```

②
```
console.log( hello.toString() );
> function() { console.log( "こんにちは" ); }
```

③
```
setTimeout( hello, 3000 );
```

④
```
setTimeout( function() { console.log( "こんにちは" ); }, 3000 );
```

練習問題　6.11　p.179

[1]　配布サンプル　to_arrow_fn.html（[1] ～ [3] 含む）

① 引数がなく、関数の本文が1行の例

引数がないため、引数の（ ）を省略できません。関数の本文は1行なので、関数の本文の{ }は省略できます。

```
const hello = () => console.log( "こんにちは" );
hello();
> こんにちは
```

② 引数が1つで、関数の本文が1行の例

引数が1つなので、引数の（ ）を省略できます。関数の本文は1行なので、関数の本文の{ }は省略できます。また、このとき関数の本文（num * 2）の実行結果は、アロー関数の戻り値になります。

```
const double = num => num * 2;
console.log( double(10) );
> 20
```

③ 無名関数を使ったコールバック関数の例

コールバック関数として引数に渡された無名関数も、今までと同様にアロー関数に書き換えることができます。

```
setTimeout( name => console.log( "こんにちは、" + name ), 3000, "独習太郎" );
> こんにちは、独習太郎
```
3秒後に以下のメッセージが表示される

この章の理解度チェック p.181

[1] 引数

① 仮引数

② 実引数

③ undefined

④ こんにちは、undefined

[2] 戻り値 **配布サンプル** sec_end2.html

① undefined

解説 return文が関数内に存在しない場合は、関数の最終行まで処理が完了した時点でundefinedが実行元（関数の呼び出し元）に返されます。

② 20

③ undefined

解説 return;のようにreturn文で値を指定しなかった場合は、undefinedが実行元に返されます。

[3] デフォルト引数

① undefined

② undefined

③ 10

④ "100"

[4] コールバック関数 **配布サンプル** sec_end4.html

① setTimeoutの利用

▶解答例

```
setTimeout( function( personName ) {
    console.log( `こんにちは、${ personName }` );
}, 2000, "太郎" );
```

② アロー関数への書き換え

▶解答例

```
setTimeout( personName => console.log( `こんにちは、${ personName }`) , 2000, "太郎" );
```

③ コールバック関数を複数利用

▶解答例

```
function add( val1, val2 ) {
    return val1 + val2;
}
function minus( val1, val2 ) {
    return val1 - val2;
}

// 3 + 2の結果がコンソールに表示される
calcAndDisp( add, console.log, 3, 2 );
> 5

calcAndDisp( minus, alert, 3, 2 );
> 1 ―――――――――――――――――――――――――― 3 - 2の結果がアラートとして画面に表示される

function calcAndDisp( calcFn, dispFn, val1, val2 ) {
    // add、minusなどがcalcFnとして実行される
    const result = calcFn( val1, val2 );
    // console.logやalertなどがdispFnとして実行される
    dispFn( result );
}
```

[5] アロー関数 配布サンプル sec_end5.html

①
```
const fn1 = ( num1, num2 ) => num1 + num2
```

②
```
const fn2 = num => num * 2
```

③
```
const fn3 = ( ) => console.log("Hello World")
```

④
```
const fn4 = name => {
    console.log( "Hello World" );
    console.log( `Hello ${name}!` );
}
```

⑤
```
const fn5 = ( ) => ( { name: "独習太郎" } )
```

第７章の解答

練習問題　7.1　p.185

［1］実行中のコードから参照できる変数や関数の範囲。

練習問題　7.2　p.194

［1］配布サンプル　which_scope.html

① a　：グローバルスコープ
② b　：スクリプトスコープ
③ fn1：グローバルスコープ
④ c　：fn1の関数スコープ
⑤ d　：fn1の関数スコープ
⑥ e　：fn1の関数スコープ　解説 varには、ブロックスコープは適用されません。
⑦ f　：ブロックスコープ
⑧ fn2：fn1の関数スコープ
⑨ fn3：スクリプトスコープ

練習問題　7.3　p.197

［1］レキシカルスコープとは、実行中のコードから見た外側のスコープのことです。

［2］配布サンプル　lexical_error_after.html

エラー発生箇所は、return result;の部分です。以下のようなエラーメッセージが表示されます。

```
> Uncaught ReferenceError: result is not defined
                                    ［意訳］参照に関するエラー：resultは定義されていません。
```

解説 resultは、return文の上のif文内で宣言されているため、ブロックスコープが適用されている状態です。そのため、ifブロックの外からは参照できないため、エラーが発生します。
このエラーを修正するには、以下のようにifブロックの外で変数resultを宣言してあげましょう。

▶解答例　修正後のコード

```
<script>
    function chance() {
        let result;
        let rand = Math.random();

        if( rand < .5 ) {
            result = "成功";
        } else {
            result = "失敗";
        }

        return result;
    }

    console.log( chance() );
</script>
```

この章の理解度チェック p.201

［1］スコープの種類

① スクリプトスコープ
② グローバルスコープ
③ 関数スコープ
④ ブロックスコープ
⑤ モジュールスコープ
⑥ 外側

［2］スコープの範囲　配布サンプル　sec_end2.html

① "関数内"

解説　スコープチェーンをたどったときに、"関数内"が最初に見つかります。

② 実行可能

解説　fn1は、レキシカルスコープ（グローバルスコープ）に位置するため、実行可能です。なお、このように関数内で再び自分自身を呼び出す関数のことを**再帰関数**と呼びます。何らかの条件で再帰関数の実行が止まるようにしていないと、無限に関数の実行が繰り返されるため、注意してください。

③ "関数内"

解説　スコープチェーンをたどったときに、"関数内"が最初に見つかります。

④ "関数内"

解説　val（"関数内"）は、スコープ内に存在するため、その値を取得します。

⑤ "グローバル"

解説　スコープチェーンをたどったときに、"グローバル"が見つかります。

⑥ "グローバル"

解説 val（"グローバル"）は、スコープ内に存在するため、その値を取得します。

⑦ "関数内"

解説 result（"関数内"）は、スコープ内に存在するため、その値を取得します。

[3] クロージャ **配布サンプル** sec_end3.html

以下は解答の一例です。実際にコードを実行して確認してみてください。

▶解答例

```
<script>
    function delayMessageFactory( printFn, ms ) {
        return function( msg ) {
            setTimeout( function() {
                printFn( msg );
            }, ms );
        };
    }
    const dialog = delayMessageFactory( alert, 2000 );
    dialog( "こんにちは" );
    // 2秒後にアラートで「こんにちは」と表示される
    const log = delayMessageFactory( console.log, 1000 );
    log( "こんばんは" );
    // 1秒後にコンソールに「こんばんは」と表示される
</script>
```

解説 関数delayMessageFactoryのprintFnとmsは、関数実行後も保持し続けられます。そのため、dialog("こんにちは")のように実行したときにも、delayMessageFactory(alert, 2000)を実行した際にprintFnとmsにそれぞれ設定されていた値（alert、2000）を保持しています。

[4] クロージャを、アロー関数を使って簡略化 **配布サンプル** sec_end4.html

次のような段階を踏んで簡略化できます。

第1段階 delayMessageFactoryをアロー関数に変更します。

```
const delayMessageFactory = (printFn, ms) =>
    function (msg) {
        setTimeout( function() {
            printFn(msg);
        }, ms);
    };
```

第2段階 delayMessageFactoryの戻り値の無名関数をアロー関数に変更します。

```
const delayMessageFactory = (printFn, ms) => msg => setTimeout(function() { printFn
(msg); }, ms);
```

第3段階 setTimeout内の無名関数をアロー関数に変更します。これで完成です。

```
const delayMessageFactory = (printFn, ms) => msg => setTimeout(() => printFn(msg),
ms);
```

解説 このように、アロー部分が連なっているときはクロージャであることが多いため、覚えておくとよいでしょう。

第8章の解答

練習問題 8.1 p.213

[1] **配布サンプル** what_this.html

① こんにちは、花子

解説 関数として実行しているため、thisはWindowオブジェクトを参照します。

② こんにちは、太郎

解説 オブジェクトのメソッドとして実行しているため、thisはtaroオブジェクトを参照します。

練習問題 8.2 p.215

[1] **配布サンプル** what_arrow_this.html

```
> 独習 太郎
```

解説 関数whichは、アロー関数のため、レキシカルスコープのthisを参照します。そして、関数whichが定義されているのはconst which = () => { ... }の部分なので、グローバルスコープになります。そのため、レキシカルスコープのthisは、Windowオブジェクトになります。この場合、関数を実行しているのは関数callNameの中ですが、関数を宣言しているのはグローバルスコープであることに注意してください。**レキシカルスコープは、関数が宣言されている位置によって決定されるスコープ**です。

練習問題　8.3　p.217

[1] 配布サンプル what_console_this.html

21

解説 コールバック関数に渡したメソッドは関数として実行されるため、Windowオブジェクトを参照します。

この章の理解度チェック　p.224

[1] 実行コンテキストとは

① グローバルコンテキスト
② 関数コンテキスト
③ this
④ コールスタック

[2] 関数コンテキストのthis　配布サンプル sec_end2.html

① こんにちは　解説 関数として実行されているため、thisはWindowオブジェクトを参照します。
② ワンワン　解説 メソッドとして実行されているため、thisはdogオブジェクトを参照します。
③ ウホウホ　解説 メソッドとして実行されているため、thisはgorillaオブジェクトを参照します。
④ こんにちは　解説 関数として実行されているため、thisはWindowオブジェクトを参照します。
⑤ こんにちは　解説 関数として実行されているため、thisはWindowオブジェクトを参照します。

[3] thisの束縛

```
setTimeout( gorilla.hello.bind( gorilla ), 2000 );
```

解説 bindでgorillaオブジェクトを束縛します。

第９章の解答

練習問題　9.1　p.234

［1］ 配布サンプル　check_roll.html

▶解答例

```
class User {
    constructor( username, password, roll ) {
        this.username = username;
        this.password = password;
        this.roll = roll;
    }

    login() {
        this.check();
        console.log( `ログイン [ ${this.username} / ${this.password} ]`);
    }

    check() {
        console.log( `ログイン情報をチェックします。`);
    }

    checkRoll() {
        if( this.roll === "admin" ) {
            console.log( "管理者権限です。" );
        } else {
            console.log( "一般ユーザーです。" );
        }
    }
}

// 一般ユーザーとしてインスタンスを生成
const taro = new User( "独習 太郎", "taro-pwd" );
taro.checkRoll();
> 一般ユーザーです。

// 管理者としてインスタンスを生成
const hanako = new User( "独習 花子", "hanako-pwd", "admin" );
hanako.checkRoll();
> 管理者権限です。
```

練習問題　9.2　p.237

[1] 配布サンプル class_error_after.html

printFnの実行文でエラーが発生しているため、printFnが定義されている場所を確認します。すると、static printFn = console.log;でエラーが発生しており、静的プロパティ（静的メソッド）としてprintFnが定義されていることがわかります。そのため、クラス名（StdClass）を先頭に付けて実行するように修正します。

▶解答例

```
class StdClass {

    constructor( arg ) {
        this.arg = arg;
    }

    static printFn = console.log;

    static print( arg ) {
       // 静的メソッド内で静的メソッドを実行する際はクラス名を付与する
       StdClass.printFn( arg );
    }

    print() {
       this.constructor.print( this.arg );
    }

 }
const std = new StdClass( "こんにちは" );
std.print();
```

練習問題　9.3　p.240

[1] 配布サンプル getter_setter_person.html

▶解答例

```
class Person {
    ... 省略

    set gender( value ) {
        if( value === "男" || value === "女"  || value === "トランスジェンダー" ) {
            this._gender = value;
        } else {
```

```
            throw new Error( 'genderプロパティには"男"、"女"、または"トランスジェンダー "を
設定してください。' );
        }
    }

    get gender() {
        // genderが取得された場合には値が保持されている _genderの値を返す
        return this._gender;
    }
}

const taro = new Person( "太郎", "独習" );
taro.gender = "男";
console.log( taro.gender );
> 男
taro.gender = "不明";    // エラーが発生！！
> genderプロパティには"男"、"女"、または"トランスジェンダー "を設定してください
```

練習問題　9.4　p.243

[1]　配布サンプル　extends_parent.html

```
class Parent {
    constructor( familyName ) {
        this.familyName = familyName;
    }
    introduction() {
        console.log( `名字は${ this.familyName }です。` );
   }
}

class Child extends Parent {
    constructor( familyName ) {
        super( familyName );
    }
}

const taro = new Child( "独習" );
taro.introduction();
> 名字は独習です。
```

練習問題　9.5　p.246

[1]　配布サンプル　has_in.html

① true　解説　自身のオブジェクトのプロパティのため、trueが返ります。

② false　解説　hasOwnPropertyメソッドでは、継承元クラスのメソッドはfalseが返ります。

③ true　解説　in演算子では、継承元クラスのメソッドはtrueが返ります。

④ true　解説　ObjectはParentが自動継承しているため、trueが返ります。

練習問題　9.6　p.250

[1]　配布サンプル　es6_to_es2022.html

▶解答例

```
class Person {

    #lastname = "独習";
    #firstname;
    #age;

    constructor( firstname ) {
        this.#firstname = firstname;
    }

    get fullname() {
        return this.#lastname + this.#firstname;
    }

    set age( value ) {
        this.#age = Number( value );
    }

    get age() {
        return this.#age;
    }

}
const taro = new Person( "太郎" );
taro.age = 18;
console.log( taro.age );
> 18
console.log( taro.fullname );
> 独習太郎
```

この章の理解度チェック p.264

[1] Userクラスの作成 配布サンプル sec_end1.html

▶解答例

```
class User {
    constructor( username ) {
        this.username = username;
        this.deleted = 0;
    }
}
```

[2] loginメソッドの実装 配布サンプル sec_end2.html

▶解答例

```
class User {
    constructor( username ) {
        this.username = username;
        this.deleted = 0;
    }
    login() {
        if( this.deleted === 0 ) {
            console.log( `${this.username}はログインに成功しました。` );
        } else {
            console.log( `${this.username}はログインに失敗しました。` );
        }
    }
}
```

[3] AdminUserクラスの作成 配布サンプル sec_end3.html

▶解答例

```
class User {
    ... 省略
}

class AdminUser extends User {
    constructor( username ) {
        super( username );
    }
    deleteUser( user ) {
        user.deleted = 1;
        console.log( `${ user.username }を削除しました。` );
    }
}
```

[4] 作成したクラスの実行　配布サンプル　sec_end4.html

▶解答例

```
class User {
    ... 省略
}
class AdminUser extends User {
    ... 省略
}

const user = new User( "独習太郎" );
const admin = new AdminUser( "独習管理者" );
admin.deleteUser( user );
> 独習太郎を削除しました。

user.login();
> 独習太郎はログインに失敗しました。
```

[5] 適切なオブジェクトかどうか判定　配布サンプル　sec_end5.html

この場合には、Userコンストラクタのインスタンスかどうかを判定するため、instanceof演算子を使います。hasOwnPropertyメソッドまたはin演算子を使ってdeletedプロパティが存在するかどうかを確認する方法もありますが、他のオブジェクトでもdeletedプロパティを保持する可能性があるため、Userオブジェクトかどうかを判定するにはhasOwnPropertyメソッドやin演算子では不適切です。

▶instanceof演算子でUserコンストラクタのインスタンスかどうかを判定

```
class User {
   ... 省略
}

class AdminUser extends User {
    constructor( username ) {
        super( username );
    }
    deleteUser( user ) {
        if( !(user instanceof User) ) {
            throw new Error( "Userオブジェクトを引数にする必要があります。" );
        }
        user.deleted = 1;
        console.log( `${ user.username }を削除しました。` );
    }
}
```

```
const user = new User( "独習太郎" );
const admin = new AdminUser( "独習管理者" );
admin.deleteUser( user );
> 独習太郎を削除しました。
admin.deleteUser( {} );    // エラーが発生！
> Userオブジェクトを引数にする必要があります。
```

最終的なコードは、以下のようになります。

▶解答例　コードの最終形

```
class User {
    constructor( username ) {
        this.username = username;
        this.deleted = 0;
    }
    login() {
        if( this.deleted === 0 ) {
            console.log( `${this.username}はログインに成功しました。` );
        } else {
            console.log( `${this.username}はログインに失敗しました。` );
        }
    }
}

class AdminUser extends User {
    constructor( username ) {
        super( username );
    }
    deleteUser( user ) {
        if( !( user instanceof User ) ) {
            throw new Error( "Userオブジェクトを引数にする必要があります。" );
        }
        user.deleted = 1;
        console.log( `${ user.username }を削除しました。` );
    }
}

const user = new User( "独習太郎" );
const admin = new AdminUser( "独習管理者" );
admin.deleteUser( user );
user.login();
```

第10章の解答

練習問題　10.1　p.271

[1] 配布サンプル interval_innerwidth.html

setIntervalで繰り返しの処理を記述し、window.innerWidthで内側境界の横幅を取得します。

▶解答例

```
setInterval( () => {
    console.log( window.innerWidth );
} , 1000 );
```

練習問題　10.2　p.277

[1] 配布サンプル whats_date.html（[1] ～ [3] 含む）

▶解答例

```
const date = new Date( "2022-05-12 03:12:13.333+09:00" );
```

[2] ▶解答例

```
const date = new Date( "2022-05-12 03:12:13.333+09:00" );

date.setDate( 15 );
console.log( date.getDay() );
> 0（日曜日）
```

[3] ▶解答例

```
date.setMonth( 7 );
console.log( date.getDay() );
> 1（月曜日）
```

練習問題　10.3　p.280

[1]　配布サンプル　advanced_date.html（[1] ～ [3] 含む）

▶解答例

```
const date= new Date( "2022-05" );

const firstDay = new Date( date.getFullYear(), date.getMonth(), 1 );
console.log( firstDay.getDay() );    // 月初
> 0（日曜日）

const lastDay = new Date( date.getFullYear(), date.getMonth() + 1, 0 );
console.log( lastDay.getDay() );    // 月末
> 2（火曜日）
```

[2]　▶解答例

```
firstDay.setDate( firstDay.getDate() + 30 );
console.log( firstDay.toDateString() );
> Tue May 31 2022
```

[3]　▶解答例

```
firstDay.setMonth( firstDay.getMonth() + 20 );
console.log( firstDay.toDateString()  );
> Wed Jan 31 2024
```

練習問題　10.4　p.287

[1]　配布サンプル　regexp_test.html

① 郵便番号に一致

▶解答例

```
function zipCodeChecker( zipcode ) {
    console.log( /^\d{3}-\d{4}$/.test( zipcode ) );
}
zipCodeChecker( "001-0012" );     // true
zipCodeChecker( "001-001" );      // false
zipCodeChecker( "2.2-3042" );     // false
zipCodeChecker( "wd3-2132" );     // false
zipCodeChecker( "124-56789" );    // false
```

② Emailに一致

▶解答例

```
function emailChecker( email ) {
    console.log( /^[\w.\-]+@[\w\-]+\.[\w\.\-]+$/.test( email ) );
}
emailChecker( "example000@gmail.com" );      // true
emailChecker( "example-0.00@gmail.com" );    // true
emailChecker( "example-0.00@ex.co.jp" );     // true
emailChecker( "example/0.00@ex.co.jp" );     // false
```

解説 [\w.\-]+は、.（ピリオド）、_（アンダーバー）、-（ハイフン）と半角英数字を表しています。

練習問題　10.5　p.292

[1] **配布サンプル** storage_tabA.html、storage_tabB.html

白い車

解説 console.log(sessionStorage.getItem("car"));の結果は、"白い車"と表示されます。これは、以下のコードで変数carの値を取得するときに、タブBのsessionStorageにはcarの値がまだ設定されていないためです。sessionStorageは、タブごとに値を保持することに注意してください。

```
const car = sessionStorage.getItem( "car" ) || localStorage.getItem( "car" );
```

そのため、localStorage.getItem("car")の結果（"白い車"）が変数carに代入され、次の行でsessionStorage.setItem("car", car);によってタブBのセッションストレージに格納されることになります。

練習問題　10.6　p.298

[1] **配布サンプル** fruit_stringify.html

① ▶解答例

```
// 変換対象のオブジェクト（文字列）
const fruits = { banana: "うまい", apple: "普通", orange: "微妙", other: { grape: "うまい" } };

console.log( JSON.stringify( fruits, [ "banana", "apple" ] ) );
> {"banana":"うまい","apple":"普通"}
```

解説 プロパティの指定なので、配列で対象のプロパティを指定します。

② ▶解答例

```
// 変換対象のオブジェクト（文字列）
const fruits = { banana: "うまい", apple: "普通", orange: "微妙", other: { grape: "うまい" } };

function replacer( key, value ) {
    if( typeof value === "string"    // valueが文字列型
            && value !== "うまい"    // valueが"うまい"以外の場合
     ) {
        return;       // JSON文字列に含まれない
     }
     return value;    // JSON文字列に含まれる
}

console.log( JSON.stringify( fruits, replacer ) );
> {"banana":"うまい","other":{"grape":"うまい"}}
```

解説 値の指定のため、replacerを使います。"うまい"以外はJSON文字列に含まないようにしましょう。なお、typeof value === "string"の記述を省略してしまうと、valueにオブジェクトが渡されたときにifブロック内に処理が入ってしまい、return;になり、オブジェクト自体をJSONに含まなくなります（そのため、結果としてundefinedになります）。replacerのvalueには、オブジェクトも渡される点に注意してください。

練習問題　10.7　p.300

[1] プリミティブ型である文字列に続けてプロパティやメソッドが記述された場合には、ラッパーオブジェクトであるStringオブジェクトのプロパティやメソッドが呼び出されるため。

この章の理解度チェック p.301

[1] Windowオブジェクト 配布サンプル end_sec1_after.html

▶解答例

```
<body style="height: 2000px; background-image: linear-gradient(#000, #fff);">
    <script>

        const intervalID = setInterval( () => {
            // ログ用
            console.log( window.scrollY );

            // スクロール量が1000pxより大きい、かつ確認ダイアログで［OK］を押したとき
            if ( window.scrollY > 1000 ) {
                if ( window.confirm( "画面を閉じますか？" ) ) {
                    window.close();
                } else {
                    clearInterval( intervalID );
                }
            }
        } , 1000 );

    </script>
</body>
```

[2] UTC、GMT、JSTとは

① 協定世界時
② GMT（グリニッジ標準時）
③ 日本標準時
④ 9（時間）

[3] 日時の計算 配布サンプル end_sec3.html

2023年5月20日深夜0時～2023年6月12日深夜0時の差分日時は、次のようにして求めることができます。

```
// ミリ秒 * 秒 * 分 * 時間 = 1日
const dayUnit = 1000 * 60 * 60 * 24;

const startDate = new Date( "2023-05-20" );
const endDate = new Date( "2023-06-12" );
const diffDays = Math.abs( endDate - startDate ) / dayUnit;
console.log( diffDays );
> 23（日）
```

[4] 正規表現 配布サンプル end_sec4_after.html

▶解答例

```
<script>
    const html = `<h1>見出し1</h1>
<h2>見出し2</h2>
<h3>見出し3</h3>
<header>ヘッダー </header>`;

    const matches = html.matchAll(/<(h[1-6])>(.+)<¥/¥1>/g);
    for (const matched of matches) {
        console.log(matched[2]);
    }
    > 見出し1
    > 見出し2
    > 見出し3
</script>
```

見出しタグは<h1>...</h1>のような形で表されるため、<h[1-6]>で<h1>から<h6>までの開始タグを表現できます。また、<(h[1-6])>のように一致したタグ名を[ ]でくくることで、終了タグに\1のように後方参照で同じタグ名を指定できます。あとは、タグにはさまれた文字列を.+ですべての文字列に一致するようにして、あとでこの部分を取得できるように( )でくくります。また、複数の一致箇所を取得する必要があるため、gフラグが必要です。

[5] ブラウザへの値の保存と復元　配布サンプル　end_sec5_after.html

▶解答例

```
<body>
    <style>
        body {
            height: 2000px;
            width: 2000px;
            background: linear-gradient(135deg, black 0%, white 100%);
        }
    </style>
    <script>
        let position = localStorage.getItem("position");
        position = JSON.parse(position);

        // positionがローカルストレージから取得できなかった場合
        if (position === null) {
            position = { x: 0, y: 0 };    // デフォルト値を設定
        } else {
            // positionが取得できた場合、スクロール
            window.scroll({ top: position.y, left: position.x, behavior: "smooth" });
        }

        // スクロールの監視とローカルストレージへの保存
        const intervalID = setInterval(() => {

            // スクロール量をpositionに保存
            position.x = window.scrollX;
            position.y = window.scrollY;

            // 値の確認
            console.log(position);

            // ローカルストレージに保存
            const json = JSON.stringify(position);
            localStorage.setItem("position", json);

        }, 1000);
    </script>
</body>
```

第11章の解答

練習問題　11.1　p.316

[1]　配布サンプル　munipulate_array.html

▶解答例

```
const chuka = [ "八宝菜", "餃子", "回鍋肉", "青椒肉絲" ];

// ① 解答　配列の末尾に"天津飯"を追加
chuka.push( "天津飯" );
> ["八宝菜", "餃子", "回鍋肉", "青椒肉絲", "天津飯"]

// ② 解答　配列の先頭に"チャーハン"を追加
chuka.unshift( "チャーハン" );
> ["チャーハン", "八宝菜", "餃子", "回鍋肉", "青椒肉絲", "天津飯"]

// ③ 解答　配列の先頭の要素を削除
chuka.shift();
> ["八宝菜", "餃子", "回鍋肉", "青椒肉絲", "天津飯"]

// ④ 解答　配列の末尾の要素を削除
chuka.pop();
> ["八宝菜", "餃子", "回鍋肉", "青椒肉絲"]

// ⑤ 解答　配列の添字が2の要素を削除
chuka.splice( 2, 1 );
> ["八宝菜", "餃子", "青椒肉絲"]

// ⑥ 解答　配列の"餃子"のインデックスを確認
console.log( chuka.indexOf( "餃子" ) );
> 1

// ⑦ 解答　配列[ "杏仁豆腐", "ごま豆腐" ]を後ろに結合
chuka = chuka.concat( [ "杏仁豆腐", "ごま豆腐" ] );
> ["八宝菜", "餃子", "青椒肉絲", "杏仁豆腐", "ごま豆腐"]

// ⑧ 解答　添字の1~3（1, 2, 3）の要素を複製
let newChuka = chuka.slice( 1, 4 );
> newChuka: ["餃子", "青椒肉絲", "杏仁豆腐"]

// ⑨ 解答　⑧で取得した配列の並びを逆に
newChuka.reverse();
> ["杏仁豆腐", "青椒肉絲", "餃子"]

// ⑩ 解答　⑧で取得した配列に"八宝菜"が含まれるかを確認
console.log( newChuka.includes( "八宝菜" ) );
> false
```

練習問題　11.2　p.328

[1]　配布サンプル　munipulate_array_callback_after.html

▶解答例

```
// { タイトル, 優先順位, 完了か否か }
// 優先順位（priority）は1：低、2：中、3:高
const todos = [
    { title: "晩御飯", priority: 2, completed: false },
    { title: "ゴミ出し", priority: 1, completed: true },
    { title: "食材の買い出し", priority: 3, completed: false },
    { title: "洗濯", priority: 2, completed: true },
    { title: "録画の視聴", priority: 1, completed: false },
];

// ① 解答　Todoリストの出力
//   完了しているタスク：「{タイトル}は完了！」
//   完了していないタスク：「{タイトル}をやらないと！」
todos.forEach( ( { title, completed } ) => {
    if( completed ) {
        console.log( `${title}は完了！` );
    } else {
        console.log( `${title}をやらないと！` );
    }
});
> 晩御飯をやらないと！
> ゴミ出しは完了！
> 食材の買い出しをやらないと！
> 洗濯は完了！
> 録画の視聴をやらないと！

// ② 解答　完了していないタスクの抽出
const notCompleted = todos.filter( ( { completed } ) => {
    return completed !== true;    // return !completed;でもOK
});

// ③ 解答　優先順位で並べ替え
notCompleted.sort( ( todoA, todoB ) => {
    return todoB.priority - todoA.priority;    // priorityを降順でソート
});

// ③ 別解　③は分割代入で記述すると次のように記述できる
notCompleted.sort( ( { priority: priorityA }, { priority: priorityB } ) => {
    return priorityB - priorityA;
});
```

```
// ④ 解答　関数（printTodo）の作成
function printTodo( todos ) {
    todos.forEach( ( { title, completed } ) => {
        if( completed ) {
            console.log( `${title}は完了！` );
        } else {
            console.log( `${title}をやらないと！` );
        }
    })
}
printTodo( notCompleted );
> 食材の買い出しをやらないと！
> 晩御飯をやらないと！
> 録画の視聴をやらないと！
```

練習問題　11.3　p.332

[1] 配布サンプル　munipulate_set.html

▶解答例

```
// ① 解答　Setを配列で初期化
const set = new Set( [ "八宝菜", "餃子", "回鍋肉", "青椒肉絲", "餃子" ] );

// ② 解答　"杏仁豆腐" と "餃子"を追加してSetオブジェクトの状態を確認
set.add( "杏仁豆腐" );
set.add( "餃子" );
console.log( set );
> {"八宝菜", "餃子", "回鍋肉", "青椒肉絲", "杏仁豆腐"}

// ③ 解答　"回鍋肉"を削除
set.delete( "回鍋肉" );

// ④ 解答　Setオブジェクトに"八宝菜"が含まれるか確認
console.log( set.has( "八宝菜" ) );
> true

// ⑤ 解答　④のSetオブジェクトを配列にして要素を結合
const arry = Array.from( set );
console.log( arry.join( " " ) );
> 八宝菜 餃子 青椒肉絲 杏仁豆腐
```

練習問題　11.4　p.338

[1]　配布サンプル　manipulate_map.html

▶解答例

```
// [ 商品名, 価格 ]
const menu = new Map( [
    [ "天津飯", 1000 ],
    [ "八宝菜",500 ],
    [ "ゴマ団子", 200 ],
] );

// ① 解答　300円の"杏仁豆腐"をメニューに追加
menu.set( "杏仁豆腐", 300 );

// ② 解答　"天津飯"の値段をコンソールに出力
console.log( menu.get( "天津飯" ) );
> 1000

// ③ 解答　"ゴマ団子"がメニューに存在するか確認
console.log( menu.has( "ゴマ団子" ) );
> true

// ④ 解答　"八宝菜"をメニューから削除
menu.delete( "八宝菜" );
```

この章の理解度チェック　p.341

[1]　配列の操作　配布サンプル　sec_end1_after.html

▶解答例

```
// [ 商品名, 個数, 金額 ]
const orders = [
    ["八宝菜", 1, 600 ],
    ["餃子", 4, 200 ],
    ["回鍋肉", 1, 500 ],
    ["青椒肉絲", 2, 700 ]
];

// ① 解答　回鍋肉を配列（orders）から除外
// 分割代入を使った場合
const orderFiltered = orders.filter( ([ itemName ]) => itemName !== "回鍋肉" );
// 分割代入を使わなかった場合
// const orderFiltered = orders.filter( order => order[ 0 ] !== "回鍋肉" );
```

```
// ② 解答　すべての商品が1000円より安いことを確認
// 分割代入を使った場合
const lt1000 = orderFiltered.every( ( [,,price] ) => price < 1000 );
// 分割代入を使わなかった場合
// const lt1000 = orderFiltered.every( order => order[ 2 ] < 1000 );
console.log( lt1000 );
> true

// ③ 解答　オーダーの金額が高いものから順にソート
// 分割代入を使った場合
orderFiltered.sort( ( [,,priceA], [,,priceB] ) => priceB - priceA );
// 分割代入を使わなかった場合
// orderFiltered.sort( ( orderA, orderB ) => orderB[ 2 ] - orderA[ 2 ] );
console.log( orderFiltered );
> [
>   ["青椒肉絲", 2, 700],
>   ["八宝菜", 1, 600],
>   ["餃子", 4, 200]
> ]

// ④ 解答　オーダーをそれぞれ出力
// 分割代入を使った場合
orderFiltered.forEach(( [ itemName, amount, price ] ) => {
    console.log( `${itemName}を${price}円で${amount}個注文しました。` )
});
// 分割代入を使わなかった場合
// orderFiltered.forEach( order => {
//     console.log( `${ order[ 0 ] }を${ order[ 2 ] }円で${ order[ 1 ] }個注文しました。` )
// });
> 青椒肉絲を700円で2個注文しました。
> 八宝菜を600円で1個注文しました。
> 餃子を200円で4個注文しました。

//  ⑤ 解答　オーダーの合計金額を出力
// 分割代入を使った場合
const reducer = ( accu, [ , amount, price] ) => accu + amount * price;
// 分割代入を使わなかった場合
// const reducer = ( accu, order ) => accu + order[ 1 ] * order[ 2 ];
const sumPrice = orderFiltered.reduce( reducer, 0 );
console.log( `合計金額は${sumPrice}円です。` );
> 合計金額は2800円です。
```

[2] 友達との関係　配布サンプル　sec_end2_after.html

▶解答例

```
// Personクラス
class Person {
    constructor( fullname, age, gender ) {
        this.fullname = fullname;
        this.age = age;
        this.gender = gender;
    }
}

// 登場人物
const taro = new Person( "太郎", 18, "男" );
const jiro = new Person( "次郎", 15, "男" );
const saburo = new Person( "三郎", 10, "男" );
const hanako = new Person( "花子", 23, "女" );
const hanayo = new Person( "花代", 18, "女" );

// 友達（friends）オブジェクト
const friends = new Map;

// ① 解答　jiroとhanayoを格納したSetオブジェクトをtaroをキーにしてfriendsに追加
friends.set( taro, new Set( [ jiro, hanayo ] ) );

// ② 解答　hanakoの友達として①と同様にhanayo, taro, saburoを追加
friends.set( hanako, new Set( [ hanayo, taro, saburo ] ) );

// ③ 解答　taroの友達としてhanakoを追加
const taroFriendSet = friends.get( taro );
taroFriendSet.add( hanako );

// ④ 解答　taroの友達を年齢順にコンソールに出力
const taroFriendArray = Array.from( taroFriendSet );
// 年齢を降順にソート
taroFriendArray.sort( ( friendA, friendB ) => friendB.age - friendA.age );
for( const person of taroFriendArray ) {
    console.log( person.fullname );
}
> 花子
> 花代
> 次郎
```

```
// ⑤ 解答　taroには異性の友達が何人いるか求める
const femaleFriends = taroFriendArray.filter( person => {
    // 同性のpersonは取り除く
    return person.gender !== taro.gender;
});
console.log( femaleFriends.length );
> 2 ―――――――――――――――――――――――――――――――――――――――――――――――― 2人

// ⑥ 解答　friendsマップにキーとして登録されている人物とその友達を一覧で出力
for( const [ person, friendSet ] of friends ) {    // ループと分割代入の併用
    let friendStr = "";
    for( const friend of friendSet ) {    // 友達が格納されたSetをループ
        friendStr += `[${friend.fullname}]`;
    }
    console.log( `私の名前は${ person.fullname }です。友達には${ friendStr }がいます。` )
}
> 私の名前は太郎です。友達には[次郎][花代][花子]がいます。
> 私の名前は花子です。友達には[花代][太郎][三郎]がいます。

// ⑥ 別解（発展）
// 分割代入やforEach、reduceなどを使った場合
friends.forEach( ( friendSet, person ) => {
    // Setを配列に変換してreduceを使う
    let friendStr = Array.from( friendSet )
       .reduce( ( accu, { fullname } ) => `${accu}[${fullname}]`, "");
   console.log( `私の名前は${ person.fullname }です。友達には${ friendStr }がいます。` )
});
```

[3] WeakMapとMapの違い

① オブジェクト

② 反復処理

③ メモリリーク

④ 弱参照

第12章の解答

練習問題　12.1　p.347

[1]　配布サンプル　range_iterator.html

▶解答例

```
function rangeIterator( min, max ) {
    let value = min;    // 初期値としてminを設定

    return {
        next() {
            if( value < max ) {    // valueがmaxより小さいとき
                return {
                    done: false,
                    value: value++    // valueを値として設定してから+1を行う
                }
            } else {
                return {
                    done: true
                }
            }
        }
    }
}

const iterator = rangeIterator( 1, 3 );
console.log( iterator.next().value );
> 1
console.log( iterator.next().value );
> 2
console.log( iterator.next().value );
> undefined
```

練習問題　12.2　p.356

[1]　配布サンプル　html_parse_generator.html

▶解答例

```
const html = `<h1>見出し1</h1>
<h2>見出し2</h2>
<h3>見出し3</h3>
<header>ヘッダー </header>`;

function* hTagGen( htmlStr ) {
    // h1~h6のマッチを確認
    const matches = html.matchAll(/<(h[1-6])>(.+)<\/\1>/g);
    for (const matched of matches) {
        // matchedの2番目にはテキストが含まれている
        yield matched[2];
    }
}

// 確認
for( const text of hTagGen( html ) ) {
    console.log( text );
}
> 見出し1
> 見出し2
> 見出し3
```

練習問題　12.3　p.359

[1]　配布サンプル　spread_opt_tax.html

▶解答例

```
function totalPrice( taxRate, ...productPrices ) {
    let sum = 0;

    for ( const price of productPrices ) {
        sum += price;
    }

    // 小数点以下の切り捨てにはMath.floor()を使う
    return Math.floor( ( 1 + taxRate / 100 ) * sum );
}

console.log( totalPrice(10, 100, 200, 300) );
> 660
console.log( totalPrice(8, 100, 200, 300, 400));
> 1080
```

練習問題　12.4　p.361

[1]　配布サンプル　spread_opt_chuka.html

▶解答例

```
const chuka = [ "回鍋肉", "青椒肉絲", "餃子" ];
const desert = [ "杏仁豆腐", "ゴマ団子" ];

// ①　解答　chukaを複製
const newChuka = [ ...chuka ];
> ["回鍋肉", "青椒肉絲", "餃子"]

// ② 解答　chukaとdesertを結合した配列を作成
const merged1 = [ ...chuka, ...desert ];
> ["回鍋肉", "青椒肉絲", "餃子", "杏仁豆腐", "ゴマ団子"]

// ③ 解答　chukaとdesertの間に"担々麺"を追加した配列を作成
const merged2 = [ ...chuka, "担々麺", ...desert ];
> ["回鍋肉", "青椒肉絲", "餃子", "担々麺", "杏仁豆腐", "ゴマ団子"]
```

この章の理解度チェック p.363

［1］イテレータの作成 配布サンプル sec_end1_after.html

以下は解答の一例です。いろいろな書き方があるため、要件を満たしていれば正解です。

▶解答例

```
function genStep( min, max, step ) {
    let currentValue = min - step;    // 初期値

    return {
        next() {
            currentValue += step;    // step分加算
            // currentValueがmaxより大きいとき、反復終了
            if ( currentValue > max ) {
                return {
                    done: true
                }
            } else {
                return {
                    done: false,
                    value: currentValue
                }
            }
        }
    }
}

const it = genStep( 4, 10, 2 );
let a = it.next();

while( !a.done ) {
    console.log( a.value );
    a = it.next();
}
> 4
> 6
> 8
> 10
```

[2] ジェネレータの作成 配布サンプル sec_end2.html

▶解答例

```
function* genStep( min, max, step ) {
    // minを初期値として、ループごとにstepを加算し、maxより大きくなったときに反復処理を終了
    for( let currentValue = min; currentValue <= max; currentValue += step ) {
        yield currentValue;
    }
}
const it = genStep( 4, 10, 2 );
let a = it.next();

while( !a.done ) {
    console.log( a.value );
    a = it.next();
}
```

[3] イテレータを使った反復可能オブジェクトの作成 配布サンプル sec_end3.html

▶解答例

```
Array.prototype[ Symbol.iterator ] = function () {
    let index = 0;
    // インスタンスにアクセスする際にはthisを使用
    let arry = this;

    return {
        next() {
            if ( index < arry.length ) {
                // インデックスが配列の長さより小さい場合には反復処理を継続
                return {
                    done: false,
                    // インデックスと値をペアで返す
                    value: [ index, arry[ index++ ] ],
                };
            } else {
                return {
                    done: true,
                };
            }
        },
    };
};
for ( let item of [ "Hello", "World" ] ) {
    console.log( item );
}
> [0, "Hello"]
> [1, "World"]
```

[4] ジェネレータを使った反復可能オブジェクトの作成　配布サンプル　sec_end4.html

▶解答例

```
Array.prototype[ Symbol.iterator ] = function* () {
    let arry = this;

    for( let index = 0; index < arry.length; index++ ) {
        // インデックスと値をペアで返す
        yield [ index, arry[ index ] ];
    }
}
for ( let item of [ "Hello", "World" ] ) {
    console.log( item );
}
> [0, "Hello"]
> [1, "World"]
```

[5] スプレッド演算子　配布サンプル　sec_end5_after.html

▶解答例

```
class Shape {
    constructor( options ) {
        const defaults = {
            type: "四角形",
            textColor: "黒",
            borderColor: "なし",
            bgColor: "白",
        };
      this.options = { ...defaults, ...options };
    }

    draw() {
        const { type, textColor, borderColor, bgColor } = this.options;
        console.log( `形:[${type}] 文字色[${textColor}] 枠色[${borderColor}]
背景色[${bgColor}]` );
    }
}

const triangle = new Shape( { type: "三角形" } );
triangle.draw();
> 形:[三角形] 文字色[黒] 枠色[なし]  背景色[白]
```

❶

解説 スプレッド演算子でオブジェクトのマージを行った際にプロパティが重複する場合には、あとから設定されたオブジェクト（options）のプロパティの値が設定されます（❶）。

第13章の解答

練習問題　13.1　p.376

[1]　配布サンプル　log_order.html

C → B → A

解説　Aは、setTimeoutの実行から1秒後に実行されます。

Bは、非同期処理のため、グローバルコンテキスト（Cの実行）の終了後に実行されます。

Cは、同期的に実行されるため、最初に実行されます。

練習問題　13.2　p.378

[1]　配布サンプル　delay.html

以下のコードでは、deleyの引数fnに対してconsole.logやalertをコールバック関数として渡しています。そのため、fn(message);とした場合には、console.log(message)のように実行されることになります。

▶解答例

```
function delay( fn, message, ms ) {
    setTimeout( function() {
        fn( message );
    }, ms );
}

// ① 解答　1秒後に「こんにちは」とコンソールに表示
delay( console.log, "こんにちは", 1000 );

// ② 解答　2秒後に「さようなら」とアラートを表示
delay( alert, "さようなら", 2000 );

// ③ 解答　 delay関数をネストして呼び出し、1秒後に「1秒経ちました。」、
// 2秒後に「さらに1秒経ちました。」とコンソールに表示
delay( function( message1 ) {

    console.log( message1 );    //「1秒経ちました。」
    // delayのコールバック関数内でさらにdelayを実行（delay関数のネスト）
    delay( function( message2 ) {

        console.log( message2 );    //「さらに1秒経ちました。」

    }, "さらに1秒経ちました。", 1000 );    // さらに1秒後

}, "1秒経ちました。", 1000 );    // 1秒後
```

練習問題　13.3　p.384

［1］ 配布サンプル promise_console.html

▶解答例

```
new Promise( ( resolve, reject ) => {

    setTimeout( () => {
        const date = new Date;    // 現在日時の取得
        const second = date.getSeconds();    // 秒数の取得

        // 2で割った余りが0、または1で条件分岐
        if( second % 2 ) {
            // 奇数
            reject( second );    // catchへ
        } else {
            // 偶数
            resolve( second );    // thenへ
        }
    }, 1000 );    // 1秒待機

}).then( second => {
    console.log( `${second}は偶数のため、成功とします。` );
}).catch( second => {
    console.error( `${second}は奇数のため、エラーとします。` );
}).finally( () => {
    console.log( "処理を終了します。" );
})
```

練習問題　13.4　p.388

[1]　配布サンプル　increment_promise.html

▶解答例

```
function promiseFactory( count ) {
    // 戻り値としてPromiseインスタンスを返す
    return new Promise( resolve => {
        setTimeout( () => {
            console.log( count );    // ログに表示
            count += 2;              // 現在のカウントに2を加算
            resolve( count );        // 次のthenを実行

        }, 1000 );
    });
}

promiseFactory( 0 )    // 0からカウントをスタート
.then( count => {  return promiseFactory( count );  } )
// .then( count => promiseFactory( count ) ) のように省略可能
.then( count => {  return promiseFactory( count );  } )
.then( count => {  return promiseFactory( count );  } );
```

練習問題　13.5　p.396

① fulfilled
② rejected
③ settled
④ fulfilled
⑤ すべて
⑥ すべて
⑦ settled

練習問題　13.6　p.398

[1]　配布サンプル　queue_order.html

A → E → D → C → B

解説　A、Eのみ同期的に実行されるため、まずA → Eの順でログが表示されます。次に、Dはジョブキュー、Cはタスクキューなので、Dのジョブから実行されます。その後、setTimeoutのコールバック関数が実行されますが、Bはジョブキューに登録されるので、さらに非同期で実行されます。そのため、Cの実行が同期的に行われてから、Bが実行されます。

練習問題　13.7　p.403

[1]　配布サンプル　make_action_after.html

▶解答例

```
function action( actionName, duration ) {
    return new Promise( resolve => {
        setTimeout( () => {
            console.log( actionName );
            resolve();
        }, duration);
    });
}

async function makeAction() {
    await action( "散歩", 500 );
    await action( "朝食", 200 );
    // Promise.allの戻り値もPromiseインスタンスなので、
    // awaitを先頭に付ければ2つのactionが完了するまで、次の処理を待機する
    await Promise.all( [ action( "昼食", 500 ), action( "おしゃべり", 100 ) ] );
    await action( "夕食", 600 );
    await action( "趣味", 400 );
}

makeAction();
> 散歩
> 朝食
> おしゃべり
> 昼食
> 夕食
> 趣味
```

練習問題　13.8　p.406

[1]　配布サンプル　fetch_daily フォルダ

▶解答例　fetch_daily/index.html

```
<script>
    async function myFetchFn() {
        // ファイルを取得
        const res = await fetch( "daily.json" );
        // JSONファイルの中身の文字列をオブジェクトに変換
        const jsonObj = await res.json();
         console.log( jsonObj.word );
    }

    // 関数を実行
    myFetchFn();
</script>
```

この章の理解度チェック　p.406

[1]　非同期処理とは

① メインスレッド
② 同期的
③ タスクキュー
④ コールスタック
⑤ イベントループ

[2]　Promiseの使い方　配布サンプル　sec_end2_after.html

① 3人でバトンをつないでリレー

▶解答例

```
function run( personName ) {
    ... 省略
}

// 結果出力用関数
const printTime = ( { personName, time } ) => console.log( `${personName}のタイムは
${time}です。` );
```

```
// リレー開始
run( "太郎" )
.then( result => {
    printTime( result );      // 太郎の結果
    return run( "次郎" );     // 次郎スタート
})
.then( result => {
    printTime( result );      // 次郎の結果
    return run( "三郎" );     // 三郎スタート
})
.then( result => {
    printTime( result );      // 三郎の結果
})
.catch( ( { personName } ) => {    // 誰かこけたとき
    console.error( `${personName}がこけました。レースやり直し！` );
});
```

② 最初にゴールした人の名前とタイムをコンソールに表示

▶解答例

```
Promise.any( [ run( "太郎" ), run( "次郎" ), run( "三郎" ) ] )
.then( ( { personName, time } ) => {
    console.log( `一番最初にゴールしたのは${personName}で、タイムは${time}です。` );
})
.catch( () => {
    console.error( "レースやり直し！" );
});
```

③ 全員がゴールしたときにそれぞれの名前とタイムをコンソールに表示

▶解答例

```
Promise.all( [ run( "太郎" ), run( "次郎" ), run( "三郎" ) ] )
.then( ( results ) => {
    for( const { personName, time } of results ) {
        console.log( `${personName}のタイムは${time}です。` );
    }
})
.catch( ( { personName } ) => {
    console.error( `${personName}がこけました。レースやり直し！` );
});
```

④ 全員がゴールまたはコケた際にそれぞれがゴールしたか、コケたかコンソールに表示

▶解答例

```
Promise.allSettled( [ run( "太郎" ), run( "次郎" ), run( "三郎" ) ] )
.then( ( results ) => {
    for( const { status, value, reason } of results ) {
        // 状態がfulfilled（成功）か判定
        if( status === "fulfilled" ) {
            console.log( `${value.personName}はゴールしました。` );
        } else {
            console.error( `${reason.personName}はコケました。` );
        }
    }
});
```

⑤ 誰かがゴールまたはコケたときにその人の名前をコンソールに表示

▶解答例

```
Promise.race( [ run( "太郎" ), run( "次郎" ), run( "三郎" ) ] )
.then( ( { personName } ) => {
    console.log( `${personName}がゴールしました。` );
}).catch( ( { personName } ) => {
    console.error( `${personName}がコケました。` );
});
```

[3] await / async 配布サンプル sec_end3.html

▶解答例

```
function run( personName ) {
    ... 省略
}

const printTime = ( { personName, time } ) => console.log( `${personName}のタイムは
${time}です。` );

// ここからがPromiseチェーンの書き換え部分
// まず、awaitを使いたいのでasync functionを定義する
async function startRelay() {
    let result;

    try {
        // run関数の戻り値はPromiseインスタンスなので、awaitを先頭に付けることで待機する
        result = await run( "太郎" );
        printTime( result );              // 太郎の結果
        result = await run( "次郎" );     // 次郎スタート
```

```
        printTime( result );              // 次郎の結果
        result = await run( "三郎" );     // 三郎スタート
        printTime( result );              // 三郎の結果

    } catch ( { personName } ) {    // 誰かこけたとき
        // Promiseのcatchメソッドはcatch節で書き換え可能
        console.error( `${personName}がこけました。レースやり直し！` );
    }
}

// リレー開始
startRelay();
```

[4] Fetch 配布サンプル sec_end4 フォルダ

▶解答例（sec_end4/index.html）

```
// JSONファイルをフェッチして、オブジェクトへの変換まで行う関数
// thenの実行結果が戻り値となるため、Promiseインスタンスが返る
function fetchJSON( file ) {
    return fetch( file ).then( res => res.json() );
}

// メイン関数
// JSONファイルを取得して、指定のログを出力する
async function main() {
    // ファイルの取得とJSONメソッドの実行
    // fruits, fruitTagsにはJSONがオブジェクトに変換された状態になる
    const fruits = await fetchJSON( "fruit.json" );
    const fruitTags = await fetchJSON( "fruit-tag.json" );

    // fruitsは配列なのでfor...ofで反復処理を記述できる
    // それぞれのオブジェクトのキー（key）と値（value）を取り出す
    for( const { key, value } of fruits ) {
        // keyでタグを取得
        const tags = fruitTags[ key ];
        // タグは配列のためjoinで文字列として結合
        const tagStr = tags.join( "," );
        // ログ出力
        console.log( `${value}は次の特徴があります。(${ tagStr })` );
    }
}

// メイン関数の実行
main();
```

第14章の解答

練習問題　14.1　p.424

［1］ 配布サンプル selector_after.html

▶解答例

```
<!DOCTYPE html>
<html>
    <head>
        <title>タイトル</title>
    </head>
    <body>
        <h1>見出し1</h1>
        <p id="idAttr" class="classAttr">
            これは段落です。<span>スパンに囲まれています。</span>
        </p>
        <p class="classAttr">これは段落です。</p>
        <input type="text" name="nameAttr" />
        <input type="password" name="pwdAttr" />
        <script>
            // 解答 ①  id属性がidAttrの要素
            document.querySelector("#idAttr");
            document.getElementById("idAttr");

            // 解答 ②  class属性がclassAttrのすべての要素
            document.querySelectorAll(".classAttr");

            // 解答 ③  すべてのpタグ要素
            document.querySelectorAll("p");

            // 解答 ④  inputタグのtype属性がtextの要素
            document.querySelector( "input[type='text']" );

            // 解答 ⑤  span要素の祖先要素でpタグの要素
            const span = document.querySelector("span");
            span.closest( "p" );

            // 解答 ⑥  h1タグの兄弟要素でtype属性がpasswordの要素
            document.querySelector("h1 ~ input[type='password']");

            // 解答 ⑦  id属性がidAttrの要素の子要素
            document.querySelector("#idAttr > *");
            // または
            document.querySelector("#idAttr").firstElementChild;
```

```
            // 解答 ⑧ inputタグのtype属性がtextの直後の要素
            document.querySelector("input[type='text'] + *");
            // または
            document.querySelector("input[type='text']").nextElementSibling;

        </script>
    </body>
</html>
```

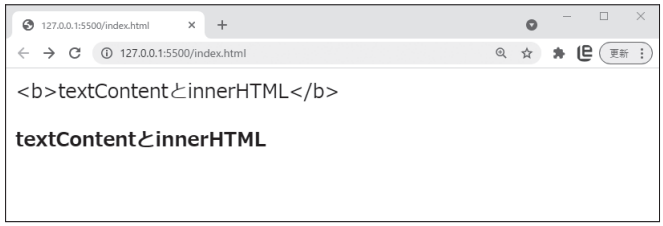
練習問題 14.2 p.429

[1] 配布サンプル textcontent_innerhtml.html

▶解答例

```
<p id="textContent"></p>
<p id="innerHTML"></p>
<script>
    const str = "<strong>textContentとinnerHTML</strong>";
    const tc = document.querySelector( "#textContent" );
    const ih = document.querySelector( "#innerHTML" );
    tc.textContent = str;
    ih.innerHTML = str;
</script>
```

実行結果 inputの場合はvalueを通して入力文字を取得・変更



練習問題 14.3 p.436

[1] 配布サンプル move_el_location_after.html

以下の解答では、Promiseチェーン用の関数moveElementにcallbackでDOM操作を行う関数を渡すことで、Promiseチェーンを作成し、2秒ごとにDOM操作を行っています。

▶解答例

```
// 2秒ごとにcallbackを実行するPromiseチェーン用関数
function moveElement( callback ) {
    // Promiseインスタンスを返す関数を戻り値とする
    return () => new Promise( resolve => {
        setTimeout(() => {
            callback();
            resolve();
        }, 2000);
    });
}

// ターゲット要素
const source = document.querySelector( "#source" );

const answer1 = moveElement(() => {
    // 解答 ① h1タグ内のspanタグの前に移動
    const h1 = document.querySelector( ".title" );
    h1.prepend( source );
});

const answer2 = moveElement(() => {
    // 解答 ② h1タグの直後に移動
    const h1 = document.querySelector( ".title" );
    h1.after( source );
});


const answer3 = moveElement(() => {
    // 解答 ③ wrapタグの子要素の末尾に移動
    const wrap = document.querySelector( ".wrap" );
    wrap.append( source );
});


const answer4 = moveElement(() => {
    // 解答 ④ liタグの2番目の文字の前に移動
    const list = document.querySelector( ".list" );
    const targetLi = list.children[ 1 ];
    targetLi.prepend( source );
});

// Promiseチェーンの実行
answer1()
.then( answer2 )
.then( answer3 )
```

```
.then( answer4 )

// ※ Promiseチェーンが難しく感じるようであれば、
//      以下のようにしても同じ挙動になります。
setTimeout( () => {
    // 解答 ①
    setTimeout( () => {
        // 解答 ②
        // ... setTimeoutのネストを解答分だけ作成
    } , 2000 );

} , 2000 );
```

練習問題　14.4　p.438

［1］ 配布サンプル change_google.html

▶解答例

```
<a href="https://google.com/">Googleへ</a>
<a href="https://www.bing.com/">Bingへ</a>
<a href="https://duckduckgo.com/">Bingへ</a>
<script>
    const links = document.querySelectorAll( "a" );

    links.forEach(link => {
        const url = link.getAttribute( "href" );
        if(/google\.com/.test(url)) {
            link.setAttribute( "href", "https://www.yahoo.co.jp/");
        }
    });

/* 以下のようにしても繰り返し処理を記述できる
for( const link of links ) {
        const url = link.getAttribute( "href" );
        if(/google\.com/.test(url)) {
            link.setAttribute( "href", "https://www.yahoo.co.jp/");
        }
    }
*/

</script>
```

練習問題　14.5　p.442

[1]　配布サンプル　bounding_rect_after.html

▶解答例

```
const target = document.querySelector("#target");
const targetRect = target.getBoundingClientRect();

// 解答 ①　ビューポートの上端から要素の枠線の上端までの距離
console.log(targetRect.y);

// 解答 ②　ビューポートの左端から要素の枠線の右端までの距離
console.log(targetRect.right);

// 解答 ③　要素の枠線（border）を含めた横幅
console.log(targetRect.width);

// 解答 ④　要素の枠線（border）を含めた縦幅
console.log(targetRect.height);
```

練習問題　14.6　p.449

[1]　配布サンプル　change_style_class.html

▶解答例

```
<div>このタグの色を変更しましょう。</div>
<style>
    .color-red {
        color: red;
    }
</style>
<script>
    const div = document.querySelector( "div" );
    div.style.backgroundColor = "gray";
    div.classList.add( "color-red" );
</script>
```

この章の理解度チェック p.449

[1] DOM

① HTML
② Document
③ DOMツリー
④ Node
⑤ Element

[2] 親子関係を表すプロパティ 配布サンプル sec_end2_after.html

▶解答例

```
<script>
    // ※それぞれの取得結果をconsole.logを使って確認してみてください。
    const me = document.querySelector("#me");
    // 解答 ①  子要素をすべて取得
    me.children;
    // 解答 ②  子要素の中の最初の要素を取得
    me.firstElementChild;
    // 解答 ③  子要素の中の最後の要素を取得
    me.lastElementChild;
    // 解答 ④  次の兄弟要素
    me.nextElementSibling;
    // 解答 ⑤  前の兄弟要素
    me.previousElementSibling;
    // 解答 ⑥  親要素
    me.parentElement;
    // 解答 ⑦  次の兄弟要素の子要素
    me.nextElementSibling.children;
    // 解答 ⑧  前の兄弟要素の子要素の中の最後の要素を取得
    me.previousElementSibling.lastElementChild;
</script>
```

[3] セレクタと要素の変更　配布サンプル　sec_end3.html

▶解答例

```
// 解答 ①  #main-titleに「タイトル」という文字列を追加
const mainTitle = document.querySelector("#main-title");
mainTitle.textContent = "タイトル";

// 解答 ②  #sub-titleに「<b>サブタイトル</b>」というHTMLを追加
const subTitle = document.querySelector(".sub-title");
subTitle.innerHTML = "<b>サブタイトル</b>";

// 解答 ③  .childの.order-1を.childの.order-3の後に移動
const me = document.querySelector("#me");
const child1 = me.children[0];
me.append(child1);

// 解答 ④  .childの.order-2の要素を複製して#meの先頭に追加
const child2 = document.querySelector(".child.order-2");
me.prepend(child2.cloneNode(true));

// 解答 ⑤  .child要素のデータ属性（data-color）をそれぞれの要素の文字色として適用
for (const child of me.children) {
    child.style.color = child.dataset.color;
}

// 解答 ⑥  #me要素の位置と大きさを取得して、#comment-bodyに以下のフォーマットで出力
const commentBody = document.querySelector("#comment-body");
commentBody.textContent =
    `#meのborderの上端とHTMLの上端の間隔は${me.offsetTop}pxです。\n` +
    `#meのborderの左端とHTMLの左端の間隔は${me.offsetLeft}pxです。\n` +
    `ビューポートの上端から#meの枠線の上端までの間隔は${me.getBoundingClientRect().y}px
です。\n` +
    `ビューポートの左端から#meの枠線の左端までの間隔は${me.getBoundingClientRect().x}px
です。\n` +
    `#meのborderを含めた高さは${me.offsetHeight}pxです。\n` +
    `#meのborderを含めた横幅は${me.offsetWidth}pxです。`;
```

[4] Todoアプリの作成準備　配布サンプル　sec_end4_after.html

▶解答例

```
<!DOCTYPE html>
<html>
    ... 省略
    <body>
        ... 省略
        <script>
            // #todo-list要素を取得
            const todoList = document.querySelector( "#todo-list" );

            // テンプレートを取得
            const todoItemTmpl = document.querySelector( "#todo-item-tmpl" );

            // テンプレートのcontent（.todo-item要素）を取得
            const todoItem = todoItemTmpl.content;

            /**
            *  上記の部分がcreateTodoItem関数外に書かれているのは、
            *  createTodoItemが複数回実行されたとしても、
            *  上記の部分は複数回実行する必要がないためです。
            *  一度テンプレートの要素を取得すれば
            *  あとはcloneNode( true );で複製すれば問題ありません。
            */

            // 解答 ① Todoアイテムを追加する関数の実装
            function createTodoItem( value ) {
                // todoItemを複製
                const newItem = todoItem.cloneNode( true );
                // 入力欄の要素を取得
                const newTitle = newItem.querySelector( ".todo-title" );
                // タイトルを設定
                newTitle.textContent = value;
                // Todoリストの末尾に追加
                todoList.append( newItem );
            }

            // テストコード
            createTodoItem( "1つ目" );
            createTodoItem( "2つ目" );
            createTodoItem( "3つ目" );
            // 画面上に上記のアイテムが追加されます。
```

```
            // 解答 ② Todoアイテムを削除する関数の実装
            function deleteTodoItem( item ) {
                // 渡された要素（item）を削除する
                item.remove();
            }

            // テストコード
            setTimeout( () => {
                deleteTodoItem( todoList.firstElementChild );    // 1つ目を削除
            }, 1000 );    // 1秒後

            // 解答 ③ Todoアイテムを完了とする関数の実装
            function completeTodoItem( item ) {
                // 渡された要素のクラスに対してcompletedを付け外しする
                item.classList.toggle( "completed" );
            }

            // テストコード
            setTimeout( () => {
                completeTodoItem( todoList.lastElementChild );    // 3つ目を完了へ
            }, 2000 );    // 2秒後

        </script>
    </body>
</html>
```


第15章の解答

練習問題　15.1　p.456

[1] 配布サンプル　eventhandler_after.html

▶解答例

```
<body>
    <div id="target"></div>
    <script>
        const target = document.querySelector( "#target" );
        target.onmouseenter = () => {
            target.style.background = "red";
        }
        target.onmouseleave = () => {
            target.style.background = "none";
        }
    </script>
```

```
    <style>
        ... 省略
    </style>
</body>
```

練習問題　15.2　p.462

［1］ 配布サンプル　hover_color_change_after.html

以下の解答では、Object.entries(element.dataset)によってデータ属性をfor...of文でループして汎用的に処理していますが、それぞれelement.addEventListener("mouseenter", ...);のような形で個別に処理してもかまいません。

▶解答例

```
<body>
    <div id="container" data-mouseenter="purple" data-mouseleave="none">
        <div id="target" data-mouseenter="green" data-mouseleave="none"></div>
    </div>
    <script>
        const container = document.querySelector( "#container" );
        const target = document.querySelector( "#target" );
        function setEvents( element ) {
            // datasetの反復可能オブジェクトを取得
            const datasets = Object.entries( element.dataset );
            // for...of文でデータ属性をループ
            for( const [ eventType, color ] of datasets ) {
                // イベントリスナにアクションを登録
                element.addEventListener( eventType, () => {
                    // 要素の背景色を変更
                    element.style.background = color;
                });
            }
        }

        setEvents( container );
        setEvents( target );
    </script>
    <style>
        ... 省略
    </style>
</body>
```

練習問題　15.3　p.468

[1]　配布サンプル　bubbling_alert_after.html

▶解答例

```
<body>
    <div id="container">
        ... 省略
    </div>
    <script>
        /* 解答 */
        const container = document.querySelector( "#container" );
        const wrapper = document.querySelector( "#wrapper" );
        const target = document.querySelector( "#target" );

        // 要素（element）にクリックイベントを登録する関数（clickAlert）
        // isCaptureをtrueとすることでキャプチャリングを有効にする
        function clickAlert( element, isCapture = false ) {
            // クリックイベントにアクションを登録
            element.addEventListener( "click", () => {
                // <span>タグの文字列を取得
                const spanText = element.firstElementChild.textContent;
                // アラートを画面に表示
                alert( `${spanText}のクリックイベントが発火しました。` );
            }, isCapture );
        }

        // バブリングの確認
        clickAlert( container );
        clickAlert( wrapper );
        clickAlert( target );
        // キャプチャリングの確認
        // clickAlert( container, true );
        // clickAlert( wrapper, true );
        // clickAlert( target, true );
    </script>
    <style>
        ... 省略
    </style>
</body>
```

練習問題　15.4　p.476

［1］ 配布サンプル　click_stoppropa_after.html

▶解答例

```
<body>
    ... 省略
    <script>
        ... 省略
        const wrapper = document.querySelector( "#wrapper" );
        ... 省略

        // バブリング
        clickDialog( container );
        clickDialog( wrapper );
        clickDialog( target );
        /* 解答 */
        // クリックイベントの伝播を停止するアクションの追加
        wrapper.addEventListener( "click", event => {
            event.stopPropagation();
        });
    </script>
    <style>
        ... 省略
    </style>
</body>
```

練習問題　15.5　p.479

［1］ 配布サンプル　not_stop_scroll_after.html

▶解答例

```
...省略...
    <script>
        // { passive: true }の状態ではpreventDefault()の実行は無視されるため、スクロールが可能。
        document.addEventListener( "wheel", ( event ) => {
            event.preventDefault();    // スクロール処理を停止できていない
        });
    </script>
...省略...
```

この章の理解度チェック p.484

［1］イベントの登録 配布サンプル sec_end1_after.html

▶解答例

```
<input id="val1" type="number" value="Ø"> + <input id="val2" type="number"
value="Ø"> = <span id="answer">Ø</span>
<script>
    // 各要素を取得
    const val1 = document.querySelector( "#val1" );
    const val2 = document.querySelector( "#val2" );
    const answer = document.querySelector( "#answer" );

    function inputHandler() {
        // val1とval2のvalue属性の値を加算する
        // val1.valueは文字列で取得されるため、Numberによって数値型に変換する
        answer.textContent = Number( val1.value ) + Number( val2.value );
    }

    val1.addEventListener( "input", inputHandler );
    val2.addEventListener( "input", inputHandler );
</script>
<style>
    input {
        width: 5Øpx;
    }
</style>
```

[2] イベントの伝播 配布サンプル sec_end2_after.html

▶解答例

```
<script>
    // 各要素を取得
    const article = document.querySelector( "article" );
    const div = document.querySelector( "div" );

    // article要素のクリックイベントにアクションを追加
    article.addEventListener( "click", () => {
        alert( "articleがクリックされました。" );
    });

    // div要素のクリックイベントにアクションを追加
    div.addEventListener( "click", event => {
        event.stopPropagation();    // イベント伝播を停止
        alert( "divがクリックされました。" );
    });
</script>
```

[3] Eventオブジェクト 配布サンプル sec_end3_after.html

Windowオブジェクトにclickイベントを登録することで、バブリングの際に必ず実行されるようにします。また、event.target要素にアクセスすることで、実際にイベントが発生した要素が取得できるため、これに対して背景色の変更を行います。

▶解答例

```
<script>
    window.addEventListener( "click", event => {
        event.target.style.background = "red";
        setTimeout(() => {
            event.target.style.background = "none";
        }, 1000);
    });
</script>
```

[4] イベントの発火 配布サンプル sec_end4.html

▶解答例

```
<button>ランダム</button>
<input id="val1" type="number" value="0"> + <input id="val2" type="number"
value="0"> = <span id="answer">0</span>
<script>
    /* [1] の解答 */
    const val1 = document.querySelector( "#val1" );
    const val2 = document.querySelector( "#val2" );
    const answer = document.querySelector( "#answer" );

    function inputHandler() {
        // 値を加算してanswer要素に描写する
        answer.textContent = Number( val1.value ) + Number( val2.value );
    }

    val1.addEventListener( "input", inputHandler );
    val2.addEventListener( "input", inputHandler );

    /* [4] の解答 */
    // 1～10までのランダムな値を取得する関数
    const rand1to10 = () => Math.floor( Math.random() * 10 + 1 );
    // ボタンを取得
    const randBtn = document.querySelector( "button" );
    // inputイベントを作成
    const inputEvent = new Event( "input" );
    // ランダムボタンにアクションを登録
    randBtn.addEventListener( "click", () => {
        // input要素に対してランダム値を設定
        val1.value = rand1to10();
        val2.value = rand1to10();
        // inputイベントの発火
        val1.dispatchEvent( inputEvent );
    });
</script>
<style>
    input {
        width: 50px;
    }
</style>
```

[5] Todoアプリの作成　配布サンプル　sec_end5_after.html

▶解答例

```
<!DOCTYPE html>
<html>
    <head>
        <title>Todoアプリ</title>
        <style>
            .completed > span {
                text-decoration: line-through;
                background-color: gray;
            }
        </style>
    </head>
    <body>
        <div id="todo-container">
            <input type="text" name="" id="create-input" />
            <button id="create-btn">追加</button>
            <ul id="todo-list">
                <!-- ここに.todo-itemを追加 -->
            </ul>
        </div>
        <template id="todo-item-tmpl">
            <li class="todo-item">
                <span class="todo-title"></span>
                <input type="button" class="delete-btn" value="削除" />
                <input type="button" class="complete-btn" value="完了" />
            </li>
        </template>
        <script>
            const EVENT_CLICK = "click";
            // 各要素の取得
            const createInput = document.querySelector("#create-input");
            const createBtn = document.querySelector("#create-btn");
            const todoList = document.querySelector("#todo-list");
            const todoItemTmpl = document.querySelector("#todo-item-tmpl");
            const todoItem = todoItemTmpl.content;

            // 解答 ①　アイテムの追加
            createBtn.addEventListener(EVENT_CLICK, () => {
                // [追加]ボタンを押すとcreateTodoItem関数が実行される
                createTodoItem(createInput.value);
            });
```

```
            function createTodoItem(value) {
                // todoItemを複製
                const newItem = todoItem.cloneNode(true);
                // テンプレート内の各要素の取得
                const newTitle = newItem.querySelector(".todo-title");
                const newdelBtn = newItem.querySelector(".delete-btn");
                const newCompBtn = newItem.querySelector(".complete-btn");

                // タイトルを設定
                newTitle.textContent = value;

                // 解答 ②  アイテムの削除
                newdelBtn.addEventListener(EVENT_CLICK, (event) => {
                    // input.delete-btnの親（li.todo-item）を削除
                    deleteTodoItem(event.target.parentElement);
                });

                // 解答 ③  アイテムの完了
                newCompBtn.addEventListener(EVENT_CLICK, (event) => {
                    // 完了のクラスを追加
                    completeTodoItem(event.target.parentElement);
                });

                // Todoリストの末尾に追加
                todoList.append(newItem);
            }

            function deleteTodoItem(item) {
                item.remove();
            }

            function completeTodoItem(item) {
                item.classList.toggle("completed");
            }
        </script>
    </body>
</html>
```

第16章の解答

練習問題　16.1　p.499

［1］ 配布サンプル export_import フォルダ

▶export_import/exports.js

```
/* 解答 ① */
// オブジェクト（obj）のエクスポート
export const obj = { console() { console.log( "オブジェクト（obj）" ) } };

// 関数（fn）のエクスポート
export function fn() { console.log( "関数（fn）" ) }

// クラス（StdClass）のエクスポート
export class StdClass {
    static console() { console.log( "クラス（StdClass）" ) }
}

// 関数をデフォルトエクスポート
export default function() {
    console.log( "デフォルトエクスポート関数" );
}
```

▶export_import/index.html

```
<script type="module">
    /* exports.jsと同じフォルダに作成*/

    /* 解答 ②  静的インポート */
    import defaultFn, { obj, fn, StdClass } from "./exports.js";
    defaultFn();
    obj.console();
    fn();
    StdClass.console();
    > デフォルトエクスポート関数
    > オブジェクト（obj）
    > 関数（fn）
    > クラス（StdClass）

    /* 解答 ③  動的インポート */
    async function init() {
        // 動的インポートで読み込み＋分割代入でプロパティを変数、関数、クラスとして抽出
        const { default: defaultFn, obj, fn, StdClass } = await import(
"./exports.js" );
        defaultFn();
        obj.console();
        fn();
        StdClass.console();
    }
    init();
    > デフォルトエクスポート関数
    > オブジェクト（obj）
    > 関数（fn）
    > クラス（StdClass）
</script>
```

この章の理解度チェック p.507

[1] モジュール

① インターフェイス
② 疎結合
③ 可読性
④ 保守性
⑤ 再利用性
⑥ ブラウザ
⑦ Node.js

[2] エクスポートとインポート 配布サンプル sec_end2 フォルダ

以下は解答の一例です。画面で確認してみてください。

▶解答例

```
<script type="module">
    // add-event.jsを静的インポート
    import addEvent from "./add-event.js";

    // 各要素を取得
    const val1 = document.querySelector("#val1");
    const val2 = document.querySelector("#val2");
    const answer = document.querySelector("#answer");

    // 関数内でawaitを使いたいので、asyncを関数宣言の先頭に付ける
    async function inputHandler() {
        // 動的インポートでモジュールを読み込み + 分割代入でadd関数を取得
        const { add } = await import( "./calc.js" );
        // #val1、#val2を加算した結果を#answerのテキストとして設定
        answer.textContent = add( val1.value, val2.value );
    }

    // addEventを使ってinputイベントにinputHandlerを登録
    addEvent( val1, "input", inputHandler );
    addEvent( val2, "input", inputHandler );
</script>
<style>
    input {
        width: 50px;
    }
</style>
```

[3] Strictモード 配布サンプル sec_end3.html

❶ `let protected = "守り";`

予約語（protected）が識別子として使用されているため。

❷ `justVariable = "ただの変数?";`

変数宣言されていない変数名に値を代入しているため。

❸ `undefined = 0;`

読み取り専用プロパティのため。

❹ `{ function fn() { }; }; fn();`

関数fnにブロックスコープが適用されるため。

第17章の解答

練習問題　17.1　p.514

[1] 配布サンプル node-testフォルダ

CDコマンドを使ってindex.jsが置いてあるディレクトリ（node-test）に移動し、nodeコマンドの実行結果として「node-test/index.jsが実行されました。」と表示されれば正解です。統合ターミナルを開いた状態によってカレントディレクトリは変わってくるため、以下のコマンドは解答の一例と考えてください。

▶解答例

```
cd node-test ―――――――――――――――――――――――――― node-testディレクトリへ移動
node ./index.js ―――――――――――――――――――――――― index.jsファイルを実行
> node-test/index.jsが実行されました。
```

練習問題　17.2　p.516

[1] 配布サンプル node-esmフォルダ

▶解答例

```
cd node-esm
node ./main.mjs
> ESMテスト
```

練習問題　17.3　p.528

[1] ▶バージョン一覧の確認

```
npm view webpack versions
```

実行結果

```
[
  '0.1.0',          '0.1.1',          '0.1.2',          '0.1.3',
  ... 省略
  '4.43.0',         '4.44.0',         '4.44.1',         '4.44.2',
  '4.45.0',         '4.46.0',         '5.0.0-alpha.0',  '5.0.0-alpha.1',
  ... 省略
  '5.37.0',         '5.37.1',         '5.38.0',         '5.38.1',
  '5.39.0',         '5.39.1',         '5.40.0'
]
```

1つ前のメジャーバージョンの最新のマイナーバージョン

執筆時点の最新のメジャーバージョンはバージョン5

▶1つ前のメジャーバージョンの最新（'4.46.0'）をインストール

```
npm install webpack@^4
```

または

```
npm install webpack@4.46.0
```

この章の理解度チェック p.545

[1] Node.jsとは

① ECMAScript
② window
③ global
④ npm
⑤ yarn

[2] パッケージ管理ソフト

```
mkdir npm-test ―――― npm-testディレクトリの作成
cd npm-test ―――― npm-testディレクトリへ移動
npm init -y ―――― 解答①
npm install vue@^2 ―――― 解答②
npm install eslint webpack -D ―――― 解答③
npm uninstall webpack ―――― 解答④
```

[3] サーバーサイドJavaScript 配布サンプル sec_end3フォルダ

① サーバーサイドJavaScriptの実装

まず、新しくテンプレートを作成する必要があるため、viewsフォルダの中にplus.ejsを作成します。このとき、val1、val2、sumに対してはパラメータを挿入します。また、クライアントJavaScriptを読み込むために<script src="/client-plus.js" defer></script>も記述しておきましょう。

▶sec_end3/views/plus.ejs

```
<!DOCTYPE html>
<html>
    <head>
        <title>値を合計するAPP</title>
    </head>
    <body>
        <!-- パラメータval1、val2、sumの挿入 -->
        <input id="val1" type="number" value="<%- val1 %>">
            + <input id="val2" type="number" value="<%- val2 %>">
            = <span id="answer"><%- sum %></span>
        <!-- client-plus.jsを読み込みます。 -->
        <script src="/client-plus.js" defer></script>
    </body>
</html>
```

次にserver.jsでURLのパスと表示するテンプレート（plus.ejs）をひもづけるため、新しくapp.get("/plus", ...)を実装します。

▶sec_end3/server.js

```
... 省略

app.get("/", (req, res) => {
    ... 省略
});

app.get("/plus", (req, res) => {
    // パラメータの取得（設定されていない場合は0を初期値とする）
    const { val1 = 0, val2 = 0 } = req.query;
    // 合計値を計算
    const sum = Number(val1) + Number(val2);
    /* テンプレートの読み込み */
    res.render("plus.ejs", { val1, val2, sum });
});

app.listen(port, () => {
    ... 省略
});
```

これで、サーバーサイドJavaScriptの実装は完了です。

② クライアントサイドJavaScriptの実装

クライアントサイドでは、#val1要素、#val2要素のinputイベントに対して合計値を#answer要素に挿入するアクションを追加します。

▶sec_end3/public/client-plus.js

```
// 関数の実行
plus();

function plus() {
    // 要素の取得
    const val1 = document.querySelector( "#val1" );
    const val2 = document.querySelector( "#val2" );
    const answer = document.querySelector( "#answer" );

    function inputHandler() {
        // #val1要素と#val2要素の入力値の合計を#answer要素に挿入
        answer.textContent = Number(val1.value) + Number(val2.value);
    }

    // アクションの追加
    val1.addEventListener( "input", inputHandler );
    val2.addEventListener( "input", inputHandler );
}
```

これで、すべての実装が完了です。画面で挙動を確認してみてください。