JavaScript実習

17

本日の内容

- 演算子
  - 被演算子と演算子
  - 算術演算子
  - 代入演算子
  - 比較演算子
  - 論理演算子
- 式と文
  - 式
  - 文
  - ブロック
  - 関数式と関数宣言
  - 引数
- 条件分岐
  - if文、switch文のおさらい
  - 三項演算子を使った条件分岐
- 演習

前回課題であった事例の紹介

1. getElementById()などを使わずに要素を操作していた例
    1. HTMLはidで指定した要素をwindowオブジェクトにプロパティとして追加してしまう特性があります。つまり、getElementById()などを使わずに要素を取得できてしまいます。
    2. どこからでも参照することができるし、値を代入することもできてしまうため、思わぬバグに巻き込まれしまうので、要素を取得する際は必ず getElementById()やquerySelector()を使うようにしましょう。

2. 配列を無理に使った例
    1. 配列は複数の値を扱うことができるので重宝するのですが、使わなければいけないということだけを意識して、値を1つだけ格納している人が数名いました。それなら変数に直接代入した方が効率的です。
    2. 配列の特徴を生かして複数の値を持たせて処理する思考に切り替えていきましょう。

演算子

演算子

▼前期の復習

- 演算処理を記号などで表現したもの
- 演算子には多くの種類がある
  - 算術演算子
  - 代入演算子
  - 比較演算子
  - 論理演算子

など

演算子と被演算子

演算子（オペレーター）には必ず演算する対象が存在します。この演算子の対象のことを被演算子（オペランド）と呼びます。

例えば、**1＋2** であれば、1と2が被演算子となります。このような2つの被演算子を扱う演算子を**二項演算子**と呼びます。

また、**1++** や **!1** の様な被演算子を1つだけ扱う演算子を**単項演算子**と呼びます。

さらに、3つの被演算子を使う特殊な**三項演算子**も存在します。

算術演算子

計算の際に用いる演算子。二項演算子のため演算子を挟んだ左右の被演算子を計算対象とします。前回の授業で学んだ通り + の場合は被演算子の型が異なる場合、動的な型付けが行われるため注意が必要。

演算子	名前	目的	例
+	加算	左辺と右辺を足す	1 + 9
-	減算	左辺より右辺の数を引く	20 - 5
*	乗算	左辺と右辺を掛ける	3 * 1
/	除算	左辺の数値を右辺で割る	10 / 5
%	剰余	左辺を右辺で割った余りを計算する	8 % 3　// 2
**	べき乗	左辺を右辺の数だけ掛ける 負の値も計算できる	3 ** 3　//27

代入演算子

変数に対して値を代入するときに使用する演算子。

演算子	名前	目的	例	意味
=	代入	左辺に右辺を代入する	x = y	x = y
+=	加算代入	左辺に左辺と右辺を足した値を代入	x += y	x = x + y
-=	減算代入	左辺と左辺と右辺を引いた値を代入	x -= y	x = x - y
*=	乗算代入	左辺に左辺と右辺をかけた値を代入	x *= y	x = x * y
/=	除算代入	左辺に左辺と右辺を割った値を代入	x /= y	x = x / y
%=	剰余代入	左辺に左辺と右辺を割った余りを代入	x %= y	x = x % y
**=	べき乗代入	左辺に左辺と右辺のべき乗を代入	x **= y	x = x ** y

インクリメントとデクリメント

被演算子に1を足す、1を引くことができる単項演算子。for文の加算式でよく用いられる。この授業ではconstを主として使っていくので活躍の場はほとんどない。

演算子	名前	目的	例
++	インクリメント	被演算子に1を足す	i++ / --i
--	デクリメント	被演算子から1を引く	i-- / --i

被演算子を左辺に置くか右辺に置くかで評価の方法が変わる。
演算子を後ろに置くと、代入してから1を足す。
演算子を前に置くと、1を足してから代入する。

```
let number = 0; // numberはインクリメントで加算されるのでletを使う
const a = number++; // numberは1になるがaには0が代入
const b = ++number; // numberが2になりbに2が代入
```

これからはインクリメントではなく、+= を使おう。

比較演算子

比較演算子は被演算子を比較してその結果がtrueであるかを判断します。被演算子には数値、文字列、論理値、オブジェクトが使用できる。

下記の例ではnum = 2とする

演算子	名前	目的	trueを返す例
==	等価	被演算子が等しければ trueを返す	2 == 2 '2' == 2 2 == num
===	厳密等価	被演算子が等しく、かつ同じ型であれば trueを返す こちらを使うことを意識する！	2 === num
!=	不等価	被演算子が等しくない場合に trueを返す	num != 3 num != '1'
!==	厳密不等価	被演算子が等しくなく、かつ /または同じ型でない場合に true を返す	num !== '2'

演算子	名前	目的	trueを返す例
>	より大きい	左辺が右辺よりも大きい場合に trueを返す	3 > 2 num > 4
>=	以上	左辺が右辺以上であれば trueを返す	3 >= '2' 2 >= 2
<	より小さい	左辺が右辺より小さい場合に trueを返す	2 < 3 '2' < 4
<=	以下	左辺が右辺以下であれば trueを返す	2 <= '2' num < = 4

```
=>    //これは演算子ではなく、アロー関数を表すので注意。まったく別もの。
```

論理演算子

論理演算子は左辺と右辺にそれぞれの演算子が持つ条件に合えばtrueを返します。判定時にfalseに変換される式は前回学んだfalsyにあたる式です。

演算子	名前	目的
&&	**論理積（AND)**	**左辺、右辺ともにtrueの場合にtrueを返し、どちらかがfalsyの場合はfalseを返す。** 論理積の意味として、true(1) * false(0)は0なのでfalseと考えるとわかりやすい。
||	**論理和（OR）**	**左辺か右辺のどちらかがtrueの場合にtrueを返し、どちらもfalsyの場合はfalseを返す。** 論理和の意味として、true(1) * false(0)は1なのでtrueと考えるとわかりやすい。 **論理和は短絡評価を行うので、左辺がfalseだった時点で右辺の評価はしない。仮に右辺にエラーが含まれていても通してしまうことを意味するので注意しよう。**
!	**論理否定（NOT）**	**かかる単一の被演算子がtrueであればfalseを返し、falseであればtrueを返す。あまのじゃく。**

演算子の優先順位

演算子には優先順位が設定されています。加算と乗算が含まれる計算式で乗算を先に計算するのはこの優先順位が関係しています。

授業で習った以外にも沢山の演算子が存在していますが下記の表を参考に優先順位の考え方を理解しましょう。

https://developer.mozilla.org/ja/docs/Web/JavaScript/Reference/Operators/Operator_Precedence#table

式と文

式と文

JSのプログラムは式と文で構成されています。今までif文などの文を利用する「命令」を勉強する機会が多かったのですが、今後JavaScriptの学習を進めて行く上で**「関数」を意識した「関数型プログラミング」を学習していきます**。

関数型プログラミングを学ぶ上で、式と文の違いを理解しておきましょう。

- 式は純粋に値だけを返す
  - 実行すると値が返ってくる
  - 文になることもできる（式文のページを参照）
  - 三項演算子
- 文は値を返さない
  - 文は命令であって、値は返さない
  - セミコロンで区切り、一文とする
  - if文やfor文などが代表的

式

JavaScriptでは「式」と「文」を区別します。

例えば下記のように 2 * 3 は式でその値は 6 です。

```
2 * 3 // 6
```

式は値を返す役割があり、変数に代入することができます。
上記の式はJavaScriptとしては正しいのですが、プログラムとしては、
変数に代入されているわけでも何でもないので、使い道がない式です。

プログラム内で利用するためには文に組み込んで行く必要があります。

また、式は「式文」という文になることもできます。
式文の理解は複雑なので興味がある人は別ページの詳細を参考にしてください。

文

式に意味を持たせるためには、文に組み込んで意味を持たせる必要があります。

```
const num = 2 * 3;
```

上記は変数numを宣言して2 * 3で初期化する文ですが、この文自体は下記の2つを行っているだけで**値は返しません**。変数numを評価することで値が返ります。

1. 変数numを宣言
2. 2 * 3を代入

consoleで上記の文を入力するとundefinedと表示されるのはそのためです。

変数は値を持たせる役割なので、文は変数に代入することができません。

文はif文やfor文などが代表的な例です。if文やfor文も変数に代入することはできません。

セミコロンの役割

文はセミコロン（;）で一文とすることは勉強しましたが、実際はJavaScriptが自動的に文の終わりを判断する機能を持っているのでセミコロンを記述しなくてプログラムは動作します。

ただ、セミコロンがないことで文の終わりの判定を誤ってしまうこともあるため、セミコロンはルールに従って書くようにしましょう。

式文（詳細）

プログラムに対して一番単純で効果がある形式を「式文」といいます。

1つの式に対してセミコロンをつけただけで表記されるもので、式文も値は持たずにundefinedを返します。

```
console.log(2 * 3);
6
< undefined
```

ただし、上記はconsole.log(2 * 3)という式文を実行したものだがconsoleに6が表示されています。この6は値ではなく、console.log(2 * 3)という式の**副作用**といい式文が返す値ではありません。他にprompt()やalert()を使っても副作用が起きます。

実際の値は次の行のundefinedで、式文としての値は破棄されているのです。

式文に意味があるのは、式に副作用がある場合のみです。下記の様な場合は副作用もなくどこにも使われない6という値を返すだけの計算でプログラム上は意味がありません。

```
2 * 3; // 6
```

ブロック文

文を { } で囲んだ部分をブロックと呼び、ブロック内には複数の文を書くことができます。ブロック文は単独でも書くことはできるのですが、多くの場合は if 文 や for 文などの構文に用いられます。

```
if( true ) {
 //文
 //文
}
```

ブロック文は文ですが、例外として文末に**セミコロンが不要**です。

ブロックスコープのサンプル varの場合

復習

この例では、1行目の変数nameが、if 文の中の変数 name によって書き換えられてしまっていることが確認できます。

var は関数スコープしか作れないため、if のブロック { } があったとしても var はスコープを生成しないので、同じスコープ内で変数を2回宣言していることになり、値が上書きされてしまいます。

```
var name = "日本電子専門学校";
if (name) {
    var name = "Webデザイン科";
    console.log("このブロック内では", name); //Webデザイン科と表示
}
console.log("グローバルでは", name) //Webデザイン科と表示
```

ブロックスコープのサンプル let / const の場合

復習

このサンプルでは const を使っているのでif文のブロックスコープが生成されています。
そのためif文からは1行目で宣言した変数nameが参照できずに、if 文の中と外での
nameの内容が異なります。

```
var name = "日本電子専門学校";
if (name) {
    const name = "Webデザイン科";
    console.log("このブロック内では", name); //Webデザイン科と表示
}
console.log("グローバルでは", name) //日本電子専門学校と表示
```

式と文の整理

- 式
  - 評価した結果を値として返すもの
- 文
  - セミコロンで終わる
  - 処理するステップで値を返さない

```
// 10という値を返す式
2 + 8 // 10

// 式の評価値を変数に代入している文
const total = 2 + 8; // undefined totalを評価する式を書いて初めて値を返す
```

関数式と関数宣言

今回学んだ式と文には関数の宣言も関わってきます。

今までは関数宣言をする方法で関数を作ってきましたが、関数を変数に代入することができる関数式という方法があります。

```
function func(){処理} //関数宣言
function(){処理} //無名関数
const func = function(){処理}; //関数式（無名関数を変数に代入）
```

関数名を用いない関数のことを「無名関数」と呼びます。

引数を使うことや処理の実行では大きな違いはありませんが、明確な違いが2つあります。

1. **関数宣言だと巻き上げが起こるが関数式では起こらない（後日詳しく）**
2. **関数式だと let / const が使えるので同名の関数は作成できない。関数宣言だと同名の関数を宣言して上書きすることができる**

関数式（関数リテラル）と関数宣言の違い

関数式とは、関数を値として、変数へ代入しています。

- **関数宣言はブロックで終わる文なので、セミコロンは不要**
- **関数式は「式」なので、変数に代入されて文になるのでセミコロンが必要**

```
const func = ...;
//変数funcを宣言して代入する値として関数を使っている

function(){処理} //式
```

※関数宣言も変数に代入することはできるが、スコープが発生するので、外から関数宣言で作成した名前では呼べない。

※Functionオブジェクトが生成するリテラル（後日）

引数

関数には()がついています。ここには引数(ひきすう)と呼ばれる、値(式)を書くことができます。

```
const func = function(x){ //xが仮引数
 console.log(x + 1);  // 仮引数と1を足している
}
func(100); // 100が仮引数に引き渡される実引数
```

関数を作る際には任意の**仮引数**を使います。

これは文の中で同じものが出てくれば関数を動かす際に**実引数**が渡されて、変数のように扱うことができます。

この引数を使いこなすことで同じ関数から様々な結果を返す事ができるようになり、プログラミングの幅が広がっていきます。

条件分岐

if 文のおさらい

if文では条件式はtrueかfalseのどちらかを生成する式を用いる。

条件式は前回学習した型の動的変換を意識してできるだけ動的変換をしない形で書けるようになることが望ましい。

```
if( 条件式 ){
 //文
}else if( 条件式 ){
 //文
}else{
 //文
}
```

一行で書く場合は { } を省略することもできる

```
if (条件式) 文;
```

switch文のおさらい

switch文は条件式の値が、caseで指定された値と一致すれば処理を実行する。

すべてのcaseと一致しなければdefaultを用意しておくことで、当てはまらなかった場合の処理を実行させることができる。

break; がない場合はそれ以降のすべての処理を実行してしまう。

```
switch (条件式) {
 case 'パターン1':
   //文
   break;
 case 'パターン2':
 case 'パターン3':
   //文 パターン2か3に当てはまれば実行
   break;
 default: //上記のパターンに当てはまらなければ
   //文
}
```

三項演算子

冒頭の演算子で触れましたが、三項を用いて評価する三項演算子が存在します。

三項演算子は条件演算子とも呼ばれ、条件分岐をとてもシンプルに記述することができます。

```
条件式 ? trueの場合の式 : falseの場合の式
```

条件分岐後は式しか書けない点に注意。if文とは考え方が違います。

試しに下記のプログラムを試してみましょう。

```
const age = 19;
const judge = age >= 20 ? '成年' : '未成年';
console.log(judge);
```

?の後の最初の文字列がtrueの処理の場合、コロンの後の文字列がfalseの場合になっています。

複数条件で書くこともできる

三項演算子も複数の条件を扱うことができます。

単純にfalseの処理の中にもう一つ処理を書いて重ねていくことで実現でき、改行することで可読性をあげたり工夫すると使いやすくなります。

```
const age = 16;
const judge =
 (age > 18) //条件
 ? '高校生以上' //trueの場合
 : (age < 6)  //falseの場合、条件
 ? '未就学児'  //trueの場合
 : (age < 12) //falseの場合、条件
 ? '小学生'    //trueの場合
 : (age < 15) //falseの場合、条件
 ? '中学生'   //trueの場合
 : '高校生'   //falseの場合、条件
console.log(judge);
```

複数の処理をしたい場合は関数式を使おう

三項演算子はtrueとfalseの場合の記述に式しか書くことができません。文（複数の処理を行いたい場合）を書きたい場合は関数式で対応しましょう。

```
num < 3 ? func1() : func2()

const func1 = function(){
 // 処理
}
const func2 = function(){
 // 処理
}
```

関数を使った書き方は他にも方法がありますので、今後の授業で関数を学ぶ際にいろいろと試してみましょう。

クイズアプリ

演習1

三択クイズが1問表示されるクイズアプリを作ってみよう。
※HTMLは編集しない

下記の要件を満たしてください。

1. クイズの問題、解答、正解を格納したデータを作る
   - オブジェクトを使ってデータを用意する { key:value, key:value ...}
   - オブジェクトの中にオブジェクトを格納することもできる{key:{key:value…},key:value...}
   - 解答はA,B,Cで答えられるようにする
   - クイズは配布したquiz.txtに問題を用意してあります。
2. 問題を表示させる
   - JavaScriptを読み込んだときに問題を表示させる
   - テンプレート文字列を使って問題を表示させる
3. 解答を入力するダイアログを開く
   - 解答ボタンを押すとprompt() で解答が入力できるようにする
4. 結果を表示する
   - 入力された解答と問題の解答が一致していればHTMLに「正解！」と表示
   - そうでなければ「不正解と表示」
   - 条件分岐では三項演算子を使うこと

演習2

演習1を改造してみよう。

1. 小文字で入力しても正解になるようにする
2. 解答の選択肢以外が入力されたら、「A、B、Cで解答してください」と表示する

演習3

演習2を改造してみよう。

- 問題のデータセットを3問分用意して、問題を回答すると次の問題が表示されるようにする。
- 問題の進行度、正解、不正解を記録しておけるようにする。
- 3問終了したら問題は表示されずに、記録していたデータを使って「3問中○問正解！」と結果表示され、「回答」ボタンが消える。
- 「クリア」ボタンを押下するとリセットされて何問目であっても最初に戻る（データがクリアされる）。

演習3ができるようになると、診断ゲームなどにも応用できますので是非チャレンジしてください。

課題の提出について

- 提出期限：木曜日中
- 木曜日はオープン実習です。みんなで教え合いながら学びを深めましょう！