JavaScript
チートシート

JavaScript まとめ① - 基本

基本ルール

JavaScriptとは
Webページに機能を追加できるプログラミング言語
HTMLやCSSを書き換えたり、操作に合わせて動きや機能を加えられる

大文字と小文字は区別される
同じ英単語であっても大文字と小文字で別物だと区別される
予約語は全て小文字で書く必要がある

予約語が存在する
言語自体で特別な意味を持つ単語のこと
変数名や関数名などの識別子として使用することができない

命令文の最後はセミコロンで区切る
セミコロン(;)は、文の終わりを示すために使用される
1つの文が終わったら必ずセミコロンをつける必要がある

JSでのCSS操作は極力控える
HTML要素の直接記述できるが、スタイルが混在し管理しにくくなる
一般的には、必要なスタイルをCSSにあらかじめ記述しておき、
JSでクラスやIDを操作することで、適用するスタイルを切り替える

文字列の表現方法
①シングルクォーテーションorダブルクオーテーションで囲む

②**テンプレートリテラル**(テンプレート文字列)←new!!
→バッククォーテーション(`)で囲み、${ } の中に変数を入れる
より直感的に文字列を表すことができる！

特徴①改行文字がそのまま改行として扱われる
特徴②文字列の中に式を埋め込むことができる

```
① "今「" + b + "」文字、あと「" + ( a - b ) + "」文字です。"
② `今「${ b }」文字、あと「${ a - b }」です。`
```

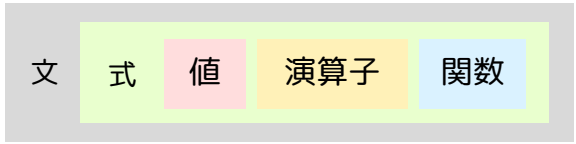
基本用語

取得
操作する対象のHTMLのタグを絞り込むこと

出力
プログラミングで入力した命令の結果を表示すること

宣言
変数や関数などの使用に先立ち、その名前などを明確にすること

呼び出し
関数や、ほかのプログラムを指定して、実行位置を移すこと



文
プログラムの中の1つの処理のこと
それ自体は値を返さないコードの部分

式
演算子、式、値、関数などを組み合わせたもの
なんらかの値を返すコードの部分

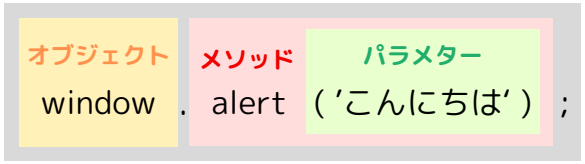
値
文字列や数値などのデータのこと

演算子
計算や比較などの演算処理を行う記号

関数
処理をまとめて定義し、再度利用できるように管理する機能

JavaScript まとめ② - 基本

基本文法



どこの　. 何をどうする (　その内容　)　　;

オブジェクト：動作させる対象となる物のこと

メソッド：命令のこと

パラメター(調整値)：メソッドを補足する情報のこと

演算子

計算や比較などの演算処理を行う記号で、結果を出すための命令のひとつ

算術演算子		代入演算子		比較演算子		理論演算子	
+	加算	=	代入	>	大なり	&&	理論積(AND)
-	減算	+=	加算代入	>=	以上	||	理論和(OR)
*	乗算	-=	減算代入	<	小なり	!	否定(NOT)
/	除算	*=	乗算代入	<=	以下		
%	剰余	/=	除算代入	==	等価		
**	累乗			===	厳密等価		
				!=(<>)	不等価		
				!==	厳密不等価		

"厳密な"とは

比較の対象にデータ型も含むかどうか
エラーの原因にもなるので、基本的には**厳密な比較を使用**する

厳密な比較　　：データ型もしっかりと比較する (ex.1と"1"は異なる)
厳密でない比較　：データ型を勝手に変換して比較する (ex.1と"1"は同じ)

データ型

データの種類のことで、基本型と参照型に分類できる

基本型(プリミティブ型)
文字や数値などの情報が入った変数
値そのものが変数に直接代入される

参照型(オブジェクト型)
配列やオブジェクトなどの場所情報が入った変数
メモリ上のアドレスが代入される

	データ型	具体例
基本型	1. **文字列型**(string)	"りんご"
	2. **数値型**(number)	13.5
	3. **長整数型**(BigInt)	52n
	3. **論理値**(boolean)	true or false
	4. **null型**	null
	5. **undefined型**	undefined
	6. **シンボル型**(symbol)	シンボル (' ')
参照型	8. **オブジェクト型**	{ key: 'value' }
	- **配列**	[1, "text", false]
	- **関数**	function name ( ) {}

cf.**エスケープシーケンス**
「\」と特定の文字を組み合わせて特殊文字を表現する
「\n」は改行を表す

JavaScript まとめ③ - 変数/定数

変数

変数とは
何らかのデータを名前を付けて保存しておく仕組みのこと
新しいデータが入ると前のデータは消える＝**１つの変数に入るデータは１つだけ**！
せっかくなら覚えさせておこう！

変数の作り方【1】
①変数を**宣言する**
②変数にデータを**代入する**
※“=”は代入するという意味

```
let 変数名；// ①
変数名 ＝“値1”；// ②
```

変数の作り方【2】
変数の宣言(①)と変数の代入(②)を
同時に行う=**変数の初期化**
こちらを使うことが多い

```
let 変数名 ＝“値1”；
```

再代入
再度値を代入して変数の値を変えること
＝箱の中身を捨てて新しい中身を入れる
再代入時、letは不要
(例)値1が上書きされ、値2に置き換わる

```
変数名 ＝“値2”；
```

定数

定数とは
しくみは変数とほぼ同じだが、定数は**一度決めた値を変更することができない**
値を代入しようとするとエラーになるので、書き換えによるバグを防ぐことができる
値を変えたくない場合もしくは**値を変える予定がない場合**に使用する
見分けがつきやすいよう**大文字で書く**ならわしがある

定数の作り方
定数を宣言する
宣言時には、**同時に値を代入**する必要がある

```
const 定数名 ＝“値1”；
```

名前を付ける際のルール
①半角英数字、_(アンダースコア)、$(ドルマーク)のみ使用できる
②数字は1文字目に使用できない、また数字のみも不可
③予約語、スペースは使えない
※ 大文字と小文字は区別されることに注意

使い分け
letは変数を宣言する時に、**constは定数を宣言**する時に使用する
constのメリットは、意図しない再代入を防げること、コードを読む時に再代入の可能性を考える必要がないこと
varは古い書き方で範囲も広くエラーの原因となるため、使用は控える
基本はconstを使用し、**再代入する予定がある時だけlet**を使用すると良い

	再宣言	再代入	スコープ	巻き上げ
const	×	×	ブロック、関数	エラー
let	×	○	ブロック、関数	エラー
var	○	○	関数	undifined

・**再宣言**
一度宣言した変数名で再度宣言すること
＝前の値を箱ごと捨てて新しい箱を用意する

・**スコープ**
変数(定数)にアクセスできる有効範囲のこと(次ページ参照)

・**巻き上げ**
変数の有効範囲が宣言を行う前にも及ぶこと(次ページ参照)

JavaScript まとめ④ - 変数/定数

スコープ

スコープとは
変数(定数)や関数を参照できる**有効範囲**のこと
変数は外に出ると生きていけない

- グローバルスコープ --------- どこからでも参照可能
- ローカルスコープ --------- 局所的な部分からのみ参照可能
  - ブロックスコープ --------- letやconstで宣言した変数(定数)や関数、if文、for文などの {} 内でのみ参照可能

・**グローバル変数**(定数) … スコープの外で宣言された変数(定数)
どこからでも参照可能
・**ローカル変数**(定数) … スコープの中で宣言された変数(定数)
そのスコープ内でのみ有効

```
let a   グローバル変数        グローバルスコープ
function ex ( ) {        ブロックスコープ
  let b  ローカル変数
} ;
                         ローカルスコープ
```

他にも…
・**モジュールスコープ**
モジュール内でのみ参照可能

・**関数スコープ**
varで宣言した変数(定数)の {} 内でのみ参照可能
ただし、var自体を使用しないためこのスコープも使用することはない

巻き上げ

巻き上げとは
関数内で宣言されたローカル変数が、その関数の先頭で宣言されたものと見なされること

```
function func() {
  console.log ( myname );   ➡ undefined と出力 ✖
  let myname = "local";
  console.log ( myname );   ➡ local と出力 ○
}
```

変数の巻き上げ

変数の宣言前に変数を参照するとエラーになる
→変数の宣言だけが先頭に移動し、まだ代入されていない状態で参照しようとしているため

巻き上げを起こさないため、関数で使用するローカル変数は**関数の先頭**で宣言すること！

cf.**値がない状態の表現** (ex.トイレットペーパーを変数に例える)

0
値が定義はされており、その値が0である状態
ex.長さが0のトイレットペーパーがセットされている

null
値の定義はされているが、値がない(空っぽの)状態
ex.トイレットペーパーがセットされてない

undefined
値が定義されていない状態
ex.トイレットペーパーホルダー自体がない

NaN
数値ではない(Not a Number)ため、返す数値が存在しない状態
ex.トイレットペーパーではないガムテープが入っている

JavaScript まとめ⑤ - 条件分岐

条件分岐

条件分岐とは
特定の条件が満たされた場合のみ処理を実行する
{ } 内は**ブロック**と呼ばれ、命令ではなく**範囲**なので**セミコロン；は付けない**

if文 … 上から順に条件を判定し、条件が合致した場合に処理を実行する

指定された条件 ( ) が真の場合、文 { } を実行する
条件が偽の場合、 else の文を実行する
else if で別の条件を加えることもできる
簡単な処理から書き、ややこしい処理は最後に残しておくと良い

```
if （条件1）{
  // 条件1が真の場合の処理;
} else if（条件2）{
  // 条件1は偽だが、条件2が真の場合の処理;
} else {
  // どの条件も偽の場合の処理;
}
```

switch文 … 特定のデータの値を判定し、条件が合致した場合に処理を実行する

特定のデータに対して複数の値を判定したい場合に使う
if分の組み合わせで代替することもできるが、if文より簡潔に書くことができる
break部分は必須、default部分は必須ではない

```
switch（式）{
  case 値1:
    // 式が値1の場合の処理;
    break;
  case 値2:
    // 式が値2の場合の処理;
    break;
  default:
  // 式がどのcaseとも一致しない場合の処理;
}
```

JavaScript まとめ⑥ - ループ処理

ループ処理

ループ処理とは
同じ処理を繰り返し実行することで、繰り返し処理や反復処理ともいう
繰り返し処理１回分を**ループ**と表現する
メリットは、コードが簡潔になり、読みやすくなること
繰り返し処理はどこかで終了させる必要がある＝**無限ループ**に注意

cf. 条件分岐と繰り返しは**制御構文**と呼ばれる
これに対し、上から下へプログラムを実行する構造を**順次**と呼ぶ
多くのプログラミング言語は**順次・条件分岐・繰り返し**の3つの構造で成り立っている

while文 … 繰り返す回数が決まっていない時

～である限り

```
while ( 条件式 ) {
  // 繰り返し実行する処理 ;
}
```

do ... while文 … 最低1回は繰り返す時

```
do {
  // 繰り返し実行する処理
} while ( 条件式 ) ;
```

※while文では1つ目の条件式が偽だった場合、処理を実行しない
do ...while文では一度処理を実行した後、繰り返すか判断をする

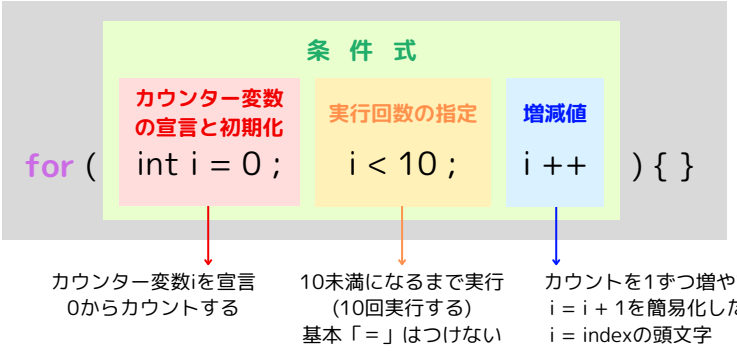
for文 … 繰り返す回数が決まっている時

～の間

```
for ( 初期化式 ; 条件式 ; 増減値 ) {
  // 繰り返し実行する処理
}
```

while文に回数をカウントするための式を付け足したもの
実際にはfor文を使うことが多い

①カウンター変数の宣言と初期化
②実行する回数(条件)
③1回の処理ごとの増減値 を記述する必要がある



cf. for (let i = 0, i < 変数名.length; i++)
＝0から始まるiの数値が、変数のデータ数と同じになるまで繰り返す

JavaScript まとめ⑦ - ループ処理

ループ処理

for ... in文 … オブジェクトから要素を取り出し順に処理する 要素を出力する

```
for ( 定数 in オブジェクト ) {
  // 繰り返し実行する処理
}
```

←「オブジェクトに属する要素を定数に順に入れる間処理を実行する」という意味

オブジェクトから取得した要素は、1つずつ定数に代入される
取得できる要素が無くなると自動的に処理は終了する
オブジェクト[定数]で要素の値を出力することもできる

for ... of文 … 配列から要素を取り出し順に処理する 値を出力する

```
for ( 定数 of 配列 ) {
  // 繰り返し実行する処理
}
```

←「配列に属する要素を定数に順に入れる間処理を実行する」という意味

配列の値が1つずつ定数へ代入されていくので、
それぞれの「値」に対して何らかの処理を実行できるのが特徴
for ... in文と異なりオブジェクトには利用できない
比較的新しい構文

※変数の値は再代入されることはないので、値の書き換えを防ぐためにも
要素の代入先には定数を使うのがおすすめ

繰り返しの処理を抜けるbreak文とcontinue文
ループ処理内で条件分岐させるようなケースで有効な方法
while文for文ともに利用でき、if文とあわせて使う

break文 … 繰り返しを中断し、次の処理へ移る

```
while ( 条件式 ) {
  // 繰り返し実行する処理
  if ( 条件式 ) {
    break ;
  }
}
```

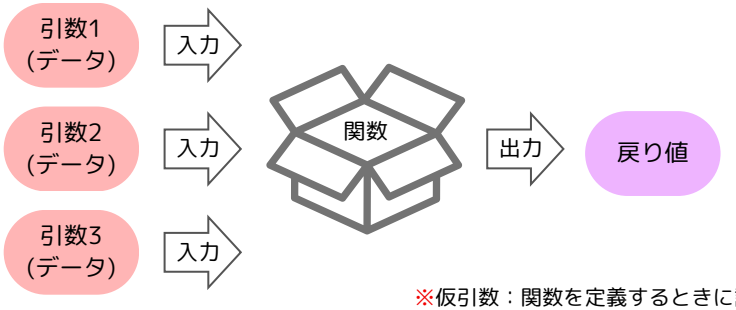
continue文 … 特定のタイミングで処理をスキップし、繰り返しを継続する

```
while ( 条件式 ) {
  // 繰り返し実行する処理
  if ( 条件式 ) {
    continue ;
  }
}
```

JavaScript まとめ⑧ - 関数

関数

関数とは
処理をまとめて定義し、再度利用できるように管理する機能
データとして受け取った引数をもとに戻り値として返す
関数名は変数と同じルールで決める
定義と呼び出しの順番はどちらでもOKだが、先に定義することが多い



※仮引数：関数を定義するときに記述する引数のこと
実引数：関数を呼び出すときに渡す引数のこと

関数のメリット
①コードが読みやすくなる(実行する処理が少なくなる)
②プログラムを再利用できる
③メンテナンス性が上がる(関数ごとに役割を分担することで修正しやすくなる)

関数を定義する方法
①**関数宣言** … 最も代表的な定義の仕方
(function文) returnと戻り値は必須ではない
returnがないと何も返却しない関数となる

```
function 関数名 ( 引数 ) {
  // 処理の内容 ;
  return 戻り値 ;
}
```

②**関数式を使用する** … **変数**(定数)**に関数を値として代入**し、後からその変数を呼び出すことで**間接的に関数を利用**する方法
(function式) イベントとセットで使うことが多い
関数名がないため、**無名関数**(匿名関数)ともいう

```
const 変数名 = function ( 引数 ) {
  // 処理の内容 ;
} ;
```

③**アロー関数式** … functionは使わず、「**=>**」(矢) を使って関数式を作る
③の関数式を簡略化するために考案されたもので、現在主流となっている
コールバック関数はアロー関数で書かれることが多い

```
const 変数名 = ( 引数 ) => {
  // 処理の内容 ;
} ;
```

※コールバック関数
呼び出さなくても実行され、ある関数(メソッド)の引数として渡す関数

・アロー関数はさらに短くできる
const 変数名 = 引数 => 処理;
・処理が一文の場合、{ }は省略可
・引数が１つの場合、( ) は省略可
・引数がない場合、( ) は必須

関数を呼び出す方法

```
関数名 ( 引数 ) ; // ①の場合
変数名 ( 引数 ) ; // ②または③の場合
```

関数とメソッドの違い
→機能は同じ！違いは**どこに所属しているか** ex. 自営業か会社員か

関数	どのオブジェクトにも所属していない一般的な関数
メソッド	関数の中でも特にオブジェクトのプロパティに入れた関数のこと

JavaScript まとめ⑨ - オブジェクト

オブジェクト

オブジェクトとは
複数のデータをひとまとまりにし、管理できるようにしたもの
キーと値がセットになった**プロパティ**を持っている
プロパティ名やメソッド名を指定して、値の取得や関数の実行ができる

値の一種なので変数やプロパティに入れることができる
＝オブジェクトのプロパティの中にさらにオブジェクトが入ることもある

変数に代入しても、直接関数の引数に記述してもOK
変数に代入した場合、変数名がオブジェクト名となる

あらかじめ用意されている組み込みオブジェクトと、開発者が独自に作成する
オブジェクトがある

※他の言語ではクラスというものに相当する

プロパティの呼び出し

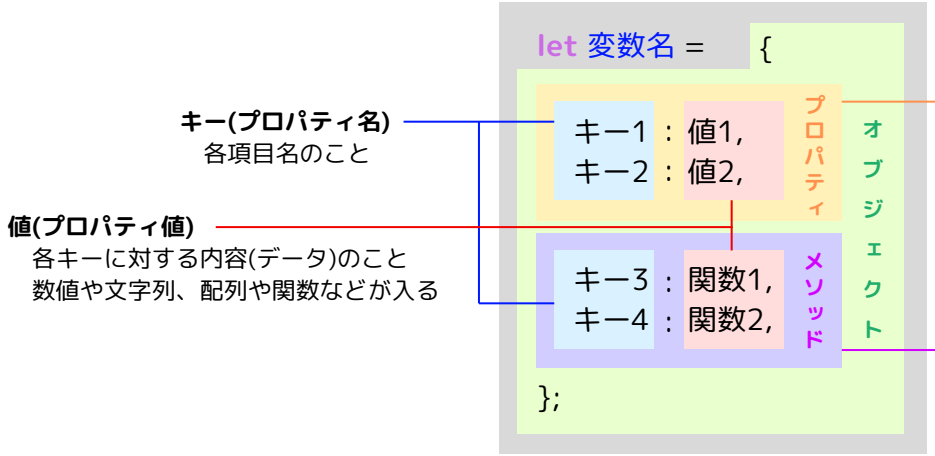
①**ドット記法** - ドットで繋げる←よく使う方法
オブジェクト.プロパティ名

②**ブラケット記法** - プロパティ名の文字列を[ ]で囲む
プロパティに変数(定数)を指定したい場合に使う
オブジェクト['プロパティ名']

メソッドの呼び出し

参照：メソッド定義の参照(実行はされない)
オブジェクト.メソッド名

実行：引数を指定し、関数と同様に実行
オブジェクト.メソッド名(引数)



キー(プロパティ名)
各項目名のこと

値(プロパティ値)
各キーに対する内容(データ)のこと
数値や文字列、配列や関数などが入る

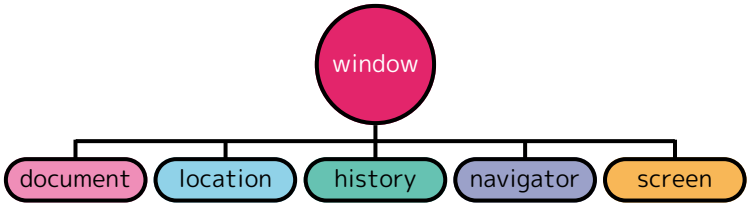
プロパティ
そのデータの種類だけが持っている情報(データ)のことで、プロパティ名と値で表される
プロパティを取得するには「オブジェクト名.プロパティ名」と記述する
プロパティ＝所有物という意味がある

メソッド
関数の中でも特にオブジェクトに属している関数のこと
メソッド名を使って値として登録された関数を呼び出すことができる
名前の後に( )が付いているものはメソッドである

JavaScript まとめ⑩ - オブジェクト

ブラウザオブジェクト

ブラウザオブジェクトとは
ブラウザの機能にアクセスするためのオブジェクトの集まりのこと
各ブラウザオブジェクトのプロパティやメソッドにアクセスしてブラウザを操作する



windowオブジェクト (グローバルオブジェクト)
ブラウザオブジェクトの階層構造の最上位に位置する
ブラウザを操作するための機能を集めたオブジェクトのこと
以下の5つのオブジェクトは全てwindowオブジェクトのプロパティである
ex.アラートを出す (alert();)

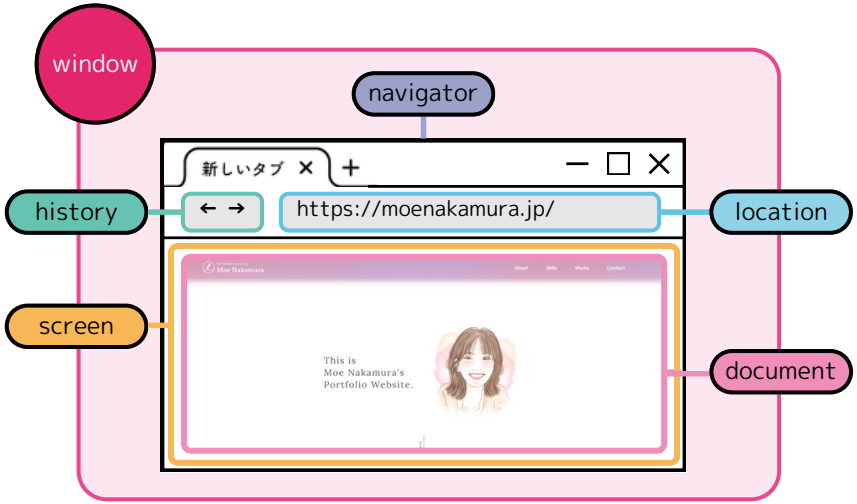
documentオブジェクト = **これがDOM！**
DOM操作ができるもの
HTMLで表現されているコンテンツを保持しているオブジェクト
HTMLの各要素を表す**elementオブジェクト**にアクセスし、HTMLを操作する
ex.要素を取得する (document.querySelector();)

locationオブジェクト
現在表示しているページのURLやアドレスに関する情報を取得できる
ex.指定したURLに移動する (location.href)

historyオブジェクト
ブラウザの履歴や、画面上に表示しているページの移動などの操作する
ex.前のページに移動する (history.back();)

navigatorオブジェクト
ブラウザのバージョンなど、ブラウザ固有の情報を提供する
ex.ブラウザの種類を知る (navigator.useAgent)

screenオブジェクト
ディスプレイに関する情報を提供する
ex.スクリーンの幅を知る (screen.width)



JavaScript まとめ⑪ - DOM

DOM

DOMとは
Document Object Modelの略
JavaScriptからHTMLにアクセスし、HTMLを操作できるようにしたもの
HTMLとJavascriptを繋ぐインターフェースである
DOMが持っているメソッドを利用してHTMLにアクセスする

DOMツリー
HTMLをツリー構造のオブジェクトとして表現したもの
documentオブジェクトからアクセスすることができる
ツリーのそれぞれの枝は**ノード**で終わる

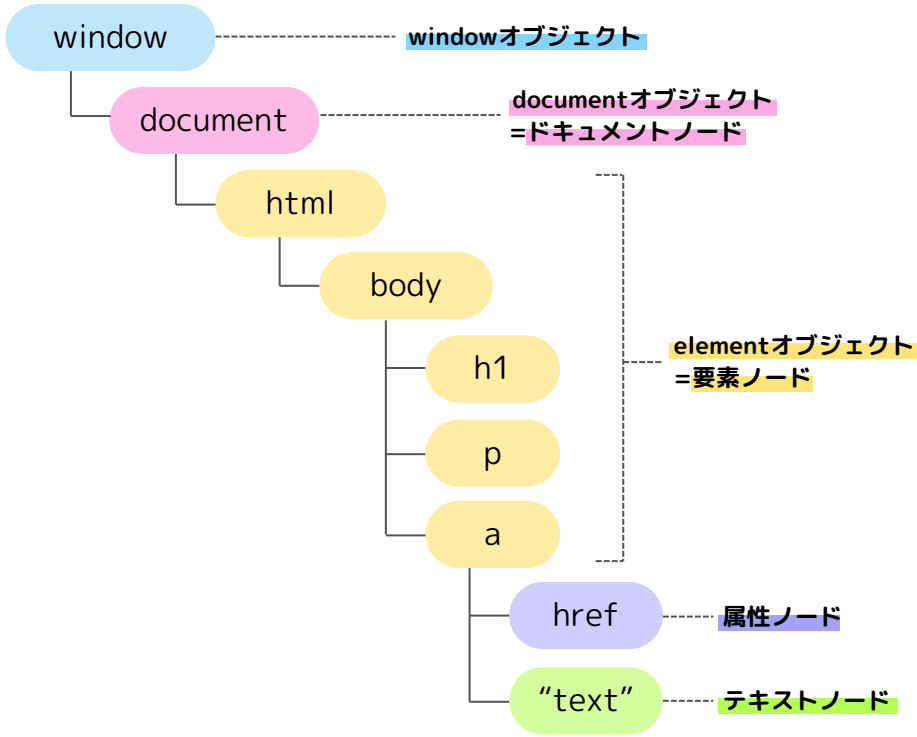
ノード
DOMツリーのパーツであり、文章を構成する個々のオブジェクトのこと
DOMツリーを操作するためのプロパティやメソッドが用意されている

- **ドキュメントノード** (document node) … documentオブジェクト
- **要素ノード** (Element node) … 要素を表すオブジェクト
- **属性ノード** (Attribute node) … 属性を表すオブジェクト
- **テキストノード** (Text node) … テキストを表すオブジェクト

Web API
Application Programming Interfaceの略
コンピューターとプログラムを繋ぐインターフェース
Webを通じて利用できるサービスと、そのサービスを利用するためのルール

elementオブジェクト
HTMLの各要素を表すオブジェクトのこと
ex.属性値を取得する (element.getAttribute();)

DOMのツリー構造



JavaScript まとめ⑫ - オブジェクト

組み込みオブジェクト

組み込みオブジェクトとは
JSに組み込まれている基本的な機能を提供するためのオブジェクトのこと
以下の3つがある

- **ビルトインオブジェクト**
  ECMAScriptにおいて、JSの標準仕様として規定されているオブジェクト
  JSの核となるもの
- **ブラウザーオブジェクト**
  Webブラウザーが独自に備えているオブジェクト
  ブラウザーによって提供される
- **DOMオブジェクト**
  HTML要素をオブジェクトとして扱えるようにするためのオブジェクト
  ブラウザーによって提供される

要素の取得

Node.textContents
要素の文字列の取得と変更を行う(HTMLは解釈しない)
タグも含めた文字列が挿入される

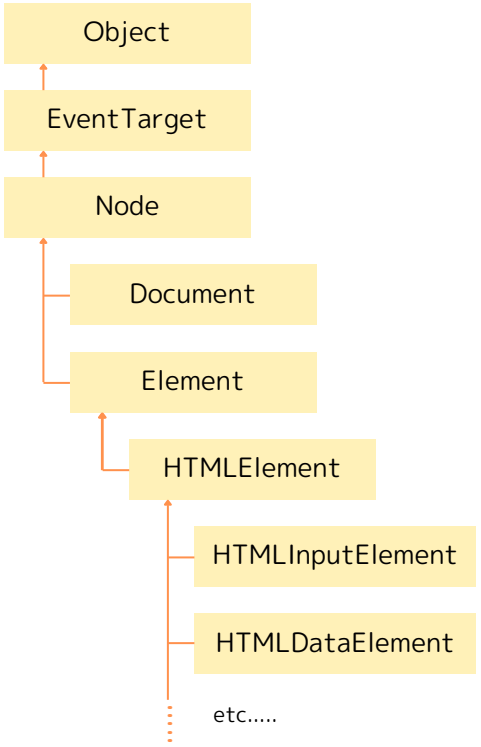
Element.innerHTML
要素の文字列の取得と変更を行う(HTMLタグを解釈する)
タグの中の文字列が挿入される

HTMLElement.innerText
要素の文字列を取得する(変更は不可)

挿入位置(insertAdjacentHTMLとinsertAdjacentElement)

```
beforebegin   要素の直前に挿入する
<div>
  afterbegin  要素内部の最初の子要素として挿入する
  text
  beforeend   要素内部の最後の子要素として挿入する
</div>
afterend      要素の直後に挿入する
```

▼Windowオブジェクト
- ▶alertメソッド
- ▶confirmメソッド
- ▶promptメソッド
- ▶parseFloatメソッド
- ▶parseIntメソッド
- ▶isNaNメソッド
- ▶setIntervalメソッド
- ▶clearIntervalメソッド
- ▶setTimeoutメソッド
- ▶clearTimeoutメソッド
- ▶Consoleオブジェクト
- ▼documentオブジェクト
  - ▶documentElementオブジェクト
- ▶Elementオブジェクト
- ▶HTMLElementオブジェクト
- ▶HTMLDataElementオブジェクト
- ▶HTMLImageElementオブジェクト
- ▶HTMLInputElementオブジェクト
- ▶Nodeオブジェクト



- ▶Arrayオブジェクト
- ▶Eventオブジェクト
- ▶EventTargetオブジェクト
- ▶IntersectionObserverオブジェクト
- ▶Stringオブジェクト

※プロパティやメソッドというからには、それが収納されているオブジェクトが必ず存在する！

JavaScript まとめ⑬ - オブジェクト

▼Windowオブジェクト

▶alertメソッド
警告ダイアログボックスを表示する　ex.**alert(‘文字列’);**

▶confirmメソッド
確認ダイアログボックスを表示する　ex.**confirm(‘文字列’);**

▶promptメソッド
入力ダイアログボックスを表示する　ex.**prompt(‘文字列’);**

▶parseFloatメソッド
文字列を実数に変換する　ex.**parseFloat(‘数値を表す文字列’);**

▶parseIntメソッド
文字列の小数点以下を切り捨て整数に変換する

ex.**parseInt(‘数値を表す文字列’);**

▶isNaNメソッド
値が非数かどうか判定する　ex.**isNaN(値);**

▶setIntervalメソッド
一定の間隔で処理を繰り返す

ex.**setInterval(処理, ミリ秒数間隔);**

※通常は使用開始と同時にタイマー識別用の変数に代入しておく

▶clearIntervalメソッド
setIntervalメソッドのタイマーを止める

ex.**clearInterval(タイマー識別用の変数);**

▶setTimeoutメソッド
一定の時間後に処理を実行する

ex.**setTimeout(処理, ミリ秒数間隔);**

※通常は使用開始と同時にタイマー識別用の変数に代入しておく

※非同期処理の関数なので、処理を待たずにその下の処理が実行される

=実行時間が不明の場合はsetIntervalよりこちらが良い

▶clearTimeOutメソッド
setTimeoutのタイマーを止める

ex.**clearTimeOut(タイマー識別用の変数);**

▼consoleオブジェクト

▶logメソッド
指定したデータをコンソールに出力する

ex.**console.log(‘ ’);**

▼documentオブジェクト

▶getElementByIdメソッド　　↓こっちで書いてOK
HTMLのid属性で要素を取得する　=　document.querySelector('#id名')

ex.**document.getElementById('id名')**

▶getElementsByClassNameメソッド
HTMLのclass属性で要素を取得する　=　document.querySelectorAll('.クラス名')

ex.**document.getElementsByClassName('クラス名')**

▶getElementsByTagNameメソッド
HTMLのタグ名で要素を取得する　=　document.querySelectorAll('タグ名')

ex.**document.getElementsByTagName('タグ名')**

▶querySelectorメソッド
指定したセレクタを持つ最初の要素を取得する

ex.**document.querySelector('CSSセレクタ')**

※複数指定は(‘.aaa .bbb .ccc’)というようにスペースで区切る

▶querySelectorAllメソッド
指定したセレクタを持つ全ての要素を取得する

ex.**document.querySelectorAll('CSSセレクタ', ...)**

▶createElementメソッド
タグのみが指定された要素を作る

ex.**document.createElement(‘タグ名’);**

▼documentElementオブジェクト

▶classListプロパティ
特定の要素のクラス名を取得する

▶addメソッド
クラスを追加する　ex.**要素.classList.add('クラス名');**

▶removeメソッド
クラスを削除する　ex.**要素.classList.remove('クラス名');**

▶replaceメソッド
指定のクラスを置換する

ex.**要素.classList.replace('対象クラス名','置換クラス名');**

▶toggleメソッド
指定クラスがあれば削除、なければ追加する

ex.**要素.classList.toggle('クラス名');**

JavaScript まとめ⑭ - オブジェクト

▼Elementオブジェクト
▶getAttributeメソッド
要素の属性の値を取得する
ex.**要素.getAttribute('属性名');**
▶setAttributeメソッド
要素の属性の値を上書きする
ex.**要素.setAttribute('属性名', '属性値');**
▶removeAttributeメソッド
要素の属性の値を削除する
ex.**要素.removeAttribute('属性名');**
▶insertAdjacentElementメソッド
指定の位置に要素を挿入する
ex.**要素.insertAdjacentElement('挿入位置', 挿入要素);**
▶insertAdjacentHTMLメソッド
指定の位置に文字列を挿入する
ex.**要素.insertAdjacentHTML('挿入位置', '挿入文字列');**
▶animateメソッド
要素を動かす　ex.**要素.animate(動かす内容, 動きの詳細);**
▶removeメソッド
要素を削除する　ex.**削除したい要素.remove();**
▶firstElementChildプロパティ
最初の子要素を取得する　ex.**要素.firstElementChild**
▶lastElementChildプロパティ
最後の子要素を取得する　ex.**要素.lastElementChild**
▶innerHTMLプロパティ
要素の文字列の取得と変更を行う(HTMLを解釈する)
ex.**要素.innerHTML = '文字列'**
▶scrollHeight/Widthプロパティ
ページ全体の高さ/幅を取得する
ex.**document.documentElement.scrollHeight/Width**
※htmlタグで作成された部分はdocument.documentElementと指定されるため
▶clientHeight/Widthプロパティ
表示領域の高さ/幅を取得する
ex.**document.documentElement.clientHeight/Width**

▼HTMLElementオブジェクト
▶clickメソッド
要素がクリックされた時に処理(関数)を実行する
ex.**要素.click(関数);**
▶innerTextプロパティ
要素の文字列を取得する(変更は不可)
ex.**要素.innerText**
▶styleプロパティ
要素のCSSスタイルの取得と変更を行う
ex.**要素.style.CSSプロパティ名 = '値';**
要素.style['CSSプロパティ名'] = '値';

▼HTMLDataElementオブジェクト
▶valueプロパティ
要素の値を取得する　ex.**要素.value**

▼HTMLImageElementオブジェクト
▶srcプロパティ
src属性を取得し、img要素に表示する画像を指定する
ex.**要素.src**
▶altプロパティ
alt属性を取得し、画像が表示されない場合のテキストを指定する
ex.**要素.alt**

▼HTMLInputElementオブジェクト
▶checkedプロパティ
チェックボックスが現在オンかオフかを取得する
ex.**要素.checked**
▶disabledプロパティ
フォーム要素のオンオフを切り替える
ex.**要素.disabled**

JavaScript まとめ⑮ - オブジェクト

▼Nodeオブジェクト
▶appendChildメソッド
親要素の末尾に子要素として追加する
ex.**親要素.appendChild(子要素);**
▶removeChildメソッド
親要素の子要素を削除する
ex.**親要素.removeChild(子要素);**
▶insertBeforeメソッド
ターゲット要素の前に親要素の子要素として挿入する
ex.**親要素.insertBefore(子要素, ターゲット要素);**
※ターゲット要素をnullにすると、最後の子要素になる
▶textContentプロパティ
要素の文字列の取得と変更を行う(HTMLは解釈しない)
ex.**要素.textContent = '文字列'**
▶parentElementプロパティ
子要素の親要素を取得する
ex.**子要素.parentElement**

▶Arrayオブジェクト
▶forEachメソッド
配列の各要素に対し、関数を順番に実行する
ex.**配列.forEach(関数 (要素の値));**
▶lengthプロパティ
配列の要素の数を取得する　ex.**配列.length**

▼Eventオブジェクト
▶targetプロパティ
イベントの呼び出し元のオブジェクトを取得する
ex.**event.target**

▼EventTargetオブジェクト
▶addEventListenerメソッド
イベント発生時、要素に対し関数を実行する
ex.**要素.addEventListener('イベント名', 関数);**

▼IntersectionObserverオブジェクト
▶IntersectionObserverメソッド
IntersectionObserverオブジェクトを作成する
指定した要素が範囲内に存在しているかを非同期で監視する
ex.**new IntersectionObserver(関数);**
▶observeメソッド
どの要素を監視するか指示する
ex.**オブジェクト.observe(要素);**

▶Mathオブジェクト
▶ceilメソッド
少数を切り上げる　ex.**Math.ceil( );**
▶floorメソッド
少数を切り捨てる　ex.**Math.floor( );**
▶randomメソッド
0以上〜1未満の数値をランダムで作る　ex.**Math.random( );**

▼Stringオブジェクト
▶replaceAllメソッド
対象文字列の中の検索文字列を置換文字列に置換する
ex.**'対象文字列'.replaceAll(検索文字列, 置換文字列);**
▶lengthプロパティ
文字数を取得する　ex.**'文字列'.length**

JavaScript まとめ⑯ - イベント

イベント

イベントとは
プログラムが動くきっかけとなるできごとのこと
マウスのクリックやページの読み込みなど、様々なアクションを指す
関数とセットで使うことが多い＝関数名をつけない無名関数
イベント発生時、あらかじめ登録しておいた処理を実行させることができる
その際に実行される処理や関数のことを**イベントハンドラ**という

①イベントに対し、処理をあらかじめ登録する
②ブラウザーがイベントの発生を監視する
③イベントの発生を検知しプログラムに通知する
④処理を呼び出し実行する

HTMLで要素の属性に直接記述する方法

```
<イベントハンドラ名 = " 関数名( ); " >
```

要素のプロパティを利用して指定する方法

```
要素 . イベントハンドラ名 = 関数名( );
```

イベントリスナーを使う方法

```
要素 . addEventListener ( ' イベント名 ' , 関数名 ) ;
```

```
要素 . addEventListener ( ' イベント名 ' , ( ) => {
  // 関数内の処理
} ) ;
```

主なイベントの種類

イベント名	イベントハンドラ名	発生するタイミング
click	onclick	クリックされた時
dblclick	ondblclick	ダブルクリックされた時
mouseover	onmouseover	マウスカーソルが重なった時
mouseout	onmouseout	マウスカーソルが離れた時
mousedown	onmousedown	マウスボタンが押された時
mouseup	onmouseup	マウスボタンを離した時
keyup	onkeyup	キーを離した時
keydown	onkeydown	キーを押した時
keypress	onkeypress	キーを押し続けている時
load	onload	ページ読み込みが完了した時
submit	onsubmit	フォームを送信する時
reset	onreset	フォームをリセットする時
select	onselect	テキストなどを選択した時
input	oninput	入力された時
change	onchange	内容が変更された時

※querySelectorAllとaddEventListenerは併用不可
代わりにforEachを使用し、そこに関数を指定する

JavaScript まとめ⑰ - 配列/this/jQuery

配列

配列とは = オブジェクトの一種！
複数の連続したデータをひとまとまりにし、管理できるようにしたもの
全体を角かっこ[ ]で囲み、値をカンマ,で区切って並べる
配列内の個々の値を**要素**といい、文字列、 数字 、変数、オブジェクトなどが入る
配列に所属するデータは前から順に**0から始まるインデックス**(番号)が与えられる
繰り返し処理と組み合わせて使われる

配列の宣言

```
let 変数名 = [ 値1, 値2, 値3 ] ;
```

※この場合、
変数名が配列名となる

配列の中の要素を取得

```
変数名 [ インデックス ] ;
```

配列の中のオブジェクトの値を取得

```
変数名 [ インデックス ] . キー ;
```

オブジェクトと配列

	格納可能データ数	管理方法	データの参照
オブジェクト	1つ	名前で管理	プロパティを使用
配列	大量	番号で管理	インデックスを使用

▸ オブジェクトin配列

```
let 変数名 = [
  { キー1 : 値1 },   ← オブジェクト
  { キー2 : 値2 },
  { キー3 : 値3 }
];                   ← 配列
```

▸ 配列inオブジェクト

```
let 変数名 = {
  キー1 : [ 値1, 値2 ],   ← 配列
  キー2 : [ 値3, 値4 ]
};                        ← オブジェクト
```

this

thisとは
JavaScriptに最初から用意されている特別な変数
プログラム内のどこでもいつでも単体で利用することができる
使用する場所により意味が異なる
何を意味するかが分かりにくくバグの原因にもなりやすいため、使用時は十分注意する

メソッド定義外で使ったthis
➡ windowオブジェクトを指す

```
console.log( this );
              ↳windowオブジェクト

// 実行結果
Window {stop: function, open:
function, alert: function…}
```

メソッド定義内で使ったthis
➡ メソッドが所属するオブジェクトを指す

```
let fortune = {
  result : [ '大吉', '中吉', '小吉' ],
  getResult : function ( ) {
    let results = this . results;
  }                    ↑
}                fortuneオブジェクト
```

※クラス定義で使用する場合、クラスを指す

jQuery

jQueryとは
JavaScriptのコードを簡潔に書くことを目的に開発されたライブラリ
ライブラリと呼ばれるプログラムの集まりのことであり、プログラミング言語ではない
現在重要度は低くなりつつある
$(セレクタ)と記述し、要素を取得する

```
$ (function( ) {
  $ (セレクタ) . メソッド (パラメター) ;
});    ↓          ↓          ↓
    どこの . 何をどうする ( その内容 ) ;
```

使い方は２つ
①ファイルをダウンロードする
②Web上から読み込む

Vanilla JS(バニラJS)
jQueryなどのライブラリを使っていない素のJavaScriptのこと