

# Webデザイン実習3B

2019/5/13

Kazuma Sekiguchi

class@cieds.jp

# 特殊な値

- undefined⇒初期化されていない変数の値

```
var a;  
alert(a);//undefined
```

- null⇒undefinedとほぼ同じだが、値を指定しないと出てこない

```
var a = null;//nullを代入  
alert(a);//null
```

- NaN⇒Not a Number。数では無いという意味だが、一般的にエラーとして出てくる

# 条件

- 条件に応じて、処理を分けるケースは多い
  - クリックされてもこれ以上右に動かさない→左に動かす
  - いわゆる判断をさせる場合に利用
  - 条件式を記述して判断させる
  - ifのみ必須
    - 他は任意
  - else ifは複数回利用可能

```
if(条件式1){
    条件式1に当てはまるならここが動作
}
else if(条件式2){
    条件式2に当てはまるならここが動作
}
else{
    条件式に当てはまらない場合に動作
}
```

# 条件式

- 0以外の数、true , 何らかの文字列は全て条件式で当てはまる
  - 0,falseだと条件式に当てはまらない
- 比較して判断することも
  - `if(a > 3)` → aが3よりも大きければtrueになる
  - `if(a == 3)` → aが3のときだけtrueになる
  - `if((a >= 3) && (a < 100))` → aが3以上で、100未満のときだけtrueになる
  - `if((a >= 100) || (a < 10))` → aが100以上または10未満のときだけtrueになる

# 条件式の成立

- 成立する場合

- `if(true)`
- `if(1以上の数値)`
- `if('何らかの文字')`
- 条件式が成立しない場合

- 不成立の場合

- `if(false)`
- `if(null)`
- `if(undefined)`
- `if(0)`
- 条件式が成立しない場合

# if文の条件

```
if(a > 5){  
  //もし変数aの値が6ならここが実行される(「成立」という)  
}  
else{  
  //もし変数aの値が5ならここが実行される(「不成立」という)  
}
```

変数aが5のとき  
if(a == 5) ⇒ 成立  
if(a > 4) ⇒ 成立  
if(a < 6) ⇒ 成立

if(a == 3) ⇒ 不成立  
if(a != 6) ⇒ 成立  
if(a == undefined) ⇒ 不成立  
if(a == null) ⇒ 不成立  
if(a == "") ⇒ 不成立

if(a > a) ⇒ 不成立  
if(a >= 5) ⇒ 成立

# オブジェクト

- JSは全ての要素がオブジェクトとして扱われる
- オブジェクト=データ及び機能の集合体（かなり強引）
- オブジェクトには何らかの値、配列、関数を格納可能
  - JSで利用している関数は全てオブジェクトに格納されている
  - オブジェクトに格納された関数のことをメソッドと呼ぶ（厳密には違う）
  - オブジェクトに格納された変数のことをプロパティと呼ぶ（厳密には違う）

# オブジェクト

- オブジェクトは自分で作成することも可能
  - 作成することで機能やデータを1つにまとめることが可能
- 作成するにはコツが要る
  - 中級くらいになると必要な場合もあるが、簡単なプログラムでは利用しなくてもプログラムは作成可能
- 利用は必ずする
  - JSで何かを作ろうとしたら必ずオブジェクトのお世話になる

# ビルトインオブジェクト

```
document.write();
```

documentオブジェクト writeメソッド

- document.write ()
  - 画面上に () 内に記述したものを記述するメソッド
  - 実際には、documentオブジェクトに存在するwriteメソッドを利用する、という記述
  - ドットで区切って考える
    - () が付けばメソッド、付かなければプロパティ

# ビルトインオブジェクト

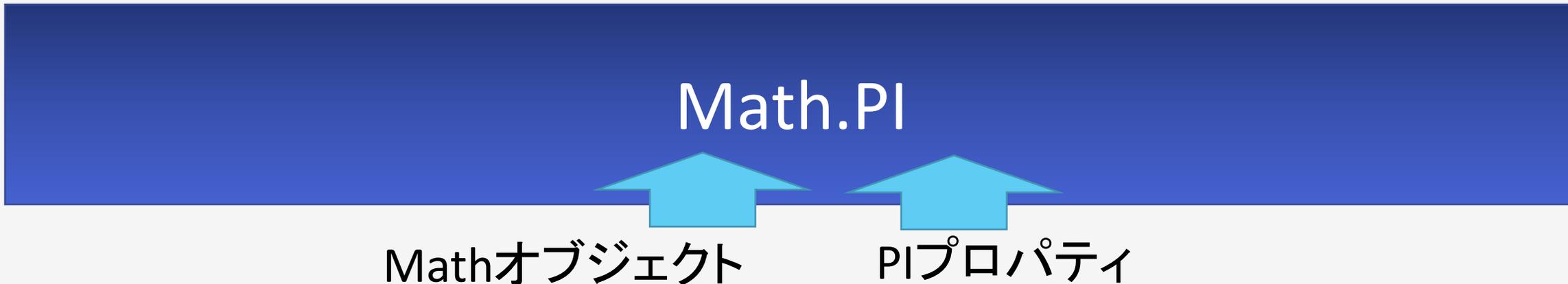
```
window.alert();
```

↑  
windowオブジェクト

↑  
alertメソッド

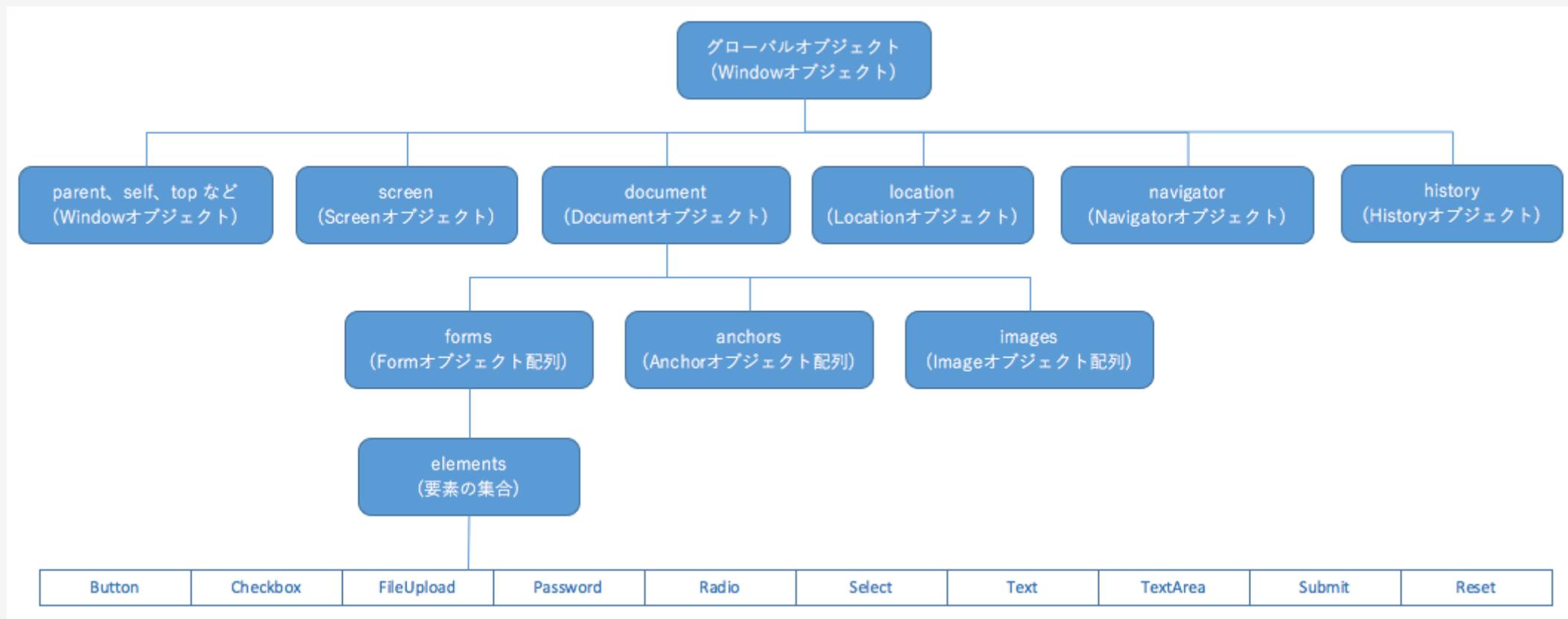
- window.alert();
  - 画面上にアラートボックスを表示する
  - 実際には、windowオブジェクトに存在するalertメソッドを利用する、という記述
- windowオブジェクトだけは表記省略が可能
  - ゆえにalert()とだけ記述すれば利用可能

# ビルトインオブジェクト



- Math.PI;
  - 円周率を取得する
  - 実際には、Mathオブジェクトに存在するPIプロパティを利用する、という記述（最後に（）が無いため、メソッドではない）

# ブラウザオブジェクト



- windowオブジェクトを頂点にした階層構造
  - 例えば、imagesオブジェクト配列を利用するのであれば、`window.document.images[0].src`のように記述して利用する
  - windowは省略可能

# ビルトインオブジェクト

- 各メソッド、プロパティはそれぞれのオブジェクトに紐付けられている
  - 利用したい機能がどのオブジェクトに属しているかは調べれば分かるので覚える必要は無い
- そのほかに組み込みオブジェクトと呼ばれるオブジェクトが存在する
  - Array , Number , String , Boolean , RegExp , Math  
Date , Function , Object  
が存在

# ビルトインオブジェクト

- ブラウザーオブジェクトと組み込みオブジェクトは使い方が少し異なる

```
document.write("JavaScript"); ← ブラウザー  
var text = "JavaScript";  
text.toUpperCase(); ← 組み込み  
document.write(text);
```

- 1行目ではJavaScriptと表示される
- 4行目ではJAVASCRIPTと表示される
- 変数に格納することで、型（文字なのか数字なのか）に沿ったメソッドを利用することが可能になる

# イベント

- イベント
  - 「何か」を「何かする」ためにした行動
- 何か = ターゲット
- 何かする = 実行するもの
- それらを繋ぐ = イベント
  
- JSは基本的にイベントに沿って実行される

# イベント

- ブラウザ上でユーザが何か操作をする
  - イベント
    - クリック、ブラウザのリサイズ、マウスを重ねた、スクロールした、時間が経過した、ページが読み込まれた、などなど
- イベントが起きたことを検知して、何か動作を起こす
  - JSはイベントが無い限り勝手に実行はされない
    - クリックしたら写真を拡大して表示
    - ブラウザをリサイズしたらレイアウトをし直す
    - マウスを重ねたら画像を変える

# イベントの種類

- クリック→onClick,onMouseDown,onMouseUp
- ダブルクリック→onDbClick
- マウスを乗せた→onMouseOver
- マウスを離れた→onMouseOut
- ページが読み込まれた→onLoad
- ページから離れた→onUnload
  
- 他にもいろいろ
- 一番良く利用されるのはクリックイベント

# イベントを紐付ける

- HTMLタグ内に属性として記述
  - 一番楽な方法
  - あまり現在では推奨されていない

```
<p onClick="show();">クリック</p>
```

- pタグをクリックしたらshow()関数を実行する
- 現在はイベントリスナーを利用するのが一般的
  - ライブラリを利用した方がイベントの紐付けが簡単

# イベントを紐付ける

- フォームからユーザーの情報を取得する
- JSを利用するとフォームからデータを取得可能

```
<form name="mainForm">  
<input type="text" name="nameData"> ← テキストフィールド  
</form>
```

テキストフィールドの  
入力内容を取得

```
document.forms.mainForm.nameData.value
```

- ボタンとイベントを紐付けて内容を取得するのが一般的

# プログラムを考える

- 時計作ろう
  - 時計 = 時間を表示させる
  - 必要なもの = 今の時間
- 今の時間を取得して、表示させる
  - 1秒に1回更新してあげれば時計の完成
- 人がやるなら
  - どこかの時計を見て、HTMLに時間を記述する
  - 1秒経過したら再度時計を見て、HTMLを更新する
  - これを繰り返す

# プログラムを考える

- プログラムの場合、人がやることをそのまま置換えて考える
  - 時間を見る = 時間を取得する
  - 時間を書く = 時間を表示する
- プログラムは基本的にバカ
  - 指定されたことしかしない
  - 何か例外なことが起きても対応できない
  - 全ての例外を考える必要がある（対応するかは別）
  - 時計の例外：止まる → 電池を交換

# JSで時間

- JSで時間を扱うためにはDateオブジェクトを利用
  - Dateオブジェクトは初期化が必要なオブジェクト
  - Mathオブジェクトは初期化が不要なオブジェクト

```
var a = Math.PI();//問題無い  
var b = Date.getHours();//エラーになる
```

- 初期化にはnewというのを利用する

```
var date = new Date();//Dateオブジェクトを初期化  
var b = date.getHours();//現在時が取得できる
```

# JSで時間

- 結構アホ
  - 時、分、秒に分けてしか取得できない
  - 年月日を取得する場合は、月のみ1少ない数を返す点に注意
- 取得できるのはPC上の時計
  - PCの時計が狂っていたら表示も狂う

```
var date = new Date();  
var h = date.getHours();//時  
var m = date.getMinutes();//分  
var s = date.getSeconds();//秒
```

# 文字列連結

- バラバラにしか取得できないので、表示するときは連結する必要がある
  - 文字列連結
  - 「+」の記号を利用するため注意
  - 両辺が数字だった場合には足し算として機能する
  - 片一方でも文字列だった場合は文字列連結になる

```
var time = h+":" +m+":" +s;
```

# タイマー

- ある一定時間毎に繰り返す際に使用

```
setTimeout(“関数名”,秒);
```

- setTimeoutで1回だけ指定した時間後に実行させることが可能
- 繰り返しであれば、setIntervalもあるが、setTimeoutは実行が保障されるのに対して、setIntervalは実行が保障されない
  - マレに動作が重いと飛ばされることがある
  - そのため現在はsetTimeoutを使う方が多い

# タイマー

- ある一定時間毎に繰り返す際に使用

```
setInterval("関数名",秒);
```

- setIntervalで指定した時間ごとに繰り返し実行させることが可能
- 終了させるためには、setIntervalの戻り値を格納しておきそれをclearIntervalで削除する

# setTimeoutで繰り返す

- setTimeoutは1回のみ指定した時間後に1度だけ実行することを保障して実行する
  - 1回で終わりなので、繰り返すには複数指定する必要がある
- setTimeoutでは関数を指定可能
  - 関数の中に関数自身を呼び出す指定を記述しておく（再帰関数）
  - それによって永久に繰り返すことが可能
- 停止したいときはif文などを挿入してsetTimeout部分を通らないようにすればOK