**初めてのJavaScript 第3版**

理解できたので復習しなくていい範囲 ＋ テストすれば再読しなくていい範囲

・1章(p1〜p6)はES6に関する説明がメインなので再読する必要無し

・2章(p7〜p22)は全体の概要を簡単に説明しているだけなので再読する必要無し

・3章(p23〜p53)のSymbolとテストだけ再確認。それ以外は再読の必要無し

・4章(p55〜p82)は制御フローの典型的な文法だけだったので、再読の必要無し

・5章(p83〜p104)はテスト部分だけ再確認すればよし 。それ以外は問題ない

・6章(p105〜p123)はオブジェクト関連のコードとthisとcall, apply, bindに関してのみ復習する

・7章 スコープ(p125〜p140)は全体的に理解できていなかった部分が多かったので、再読する必要アリ

・8章 配列(p141〜p159)は後半のmapやreduceが上手く理解できなかったので再読

・9章 オブジェクト指向(p161〜p182)は全体的に理解できたが、継承とプロトタイプの違いなどが混じってるので再読

・10章 マップとセット(p183〜p189)はイテレーションを学んだら再読

・11章 例外とエラー処理(p191〜196)は基礎的な文法を参照しているだけなので再読の必要は無し。必要になったら確認する。

・12章 イテレータとジェネレータ(p197〜p209)は再読

・13章 関数の様々な利用方法(p211〜p228)はp218の後半からのコードが分からなかったので、そこだけ再読

・14章 非同期プログラミング(p229〜p256)は全体的に分からなかったので再読するべし

・15章 日時(p257〜p269)はメソッドのような公式的側面が多かったので再読の必要無し。

・16章 Math(p271〜p278)はメソッドのような公式的側面が多かったので再読の必要無し。

・17章 正規表現(p279〜303)は

**３章　変数、定数、リテラル、データ型**

テスト

Q. プリミティブ型とは？ (p29)

A. 不変である型。数値の5は常に数値の５である

Q. undefinedとnullとNaNの違いは？

A. undefined ← まだ値を指定されていないことを表す。

　 null ← 変数の値として利用できるデータ型

　 NaN ← 数値のデータ型の一つで、数値でないことを表す

テンプレートリテラル

let currentTemp = 19.5;

const message = `現在の温度は${currentTemp}です。`;

console.log(message);

変数名や定数名として使われる文字列を識別子。

プログラムで値を直接指定するものをリテラル。

プリミティブ型とオブジェクト型しかない。

Symbol

**４章　制御フロー**

do whileループはwhileループと違って、ループ本体の処理を「少なくとも一回は」実行することが分かっている場合に使う

制御フローの例外

・break文 ループを途中で抜ける

・continue文 ループ内の今回分の処理を終了して次回分の繰り返し処理に進む

・return 文 現在の関数を終了する

・throw 文 例外ハンドラで補足する必要がある「例外」を示す

テスト

・forループパターン、p78のfor in ループとfor of ループの違いは？

for in ループ ← オブジェクトのキーに対してループする

for of ループ ← ある集合の各要素に対してループする(特に配列)

**５章　式と演算子**

JavaScriptは倍精度浮動小数点数であるから、2進数で考える。0.1などは循環小数になってしまうため、forループなどで加減乗除しても比較できない可能性がある

真と見なされる値と偽と見なされる値

真

・任意のオブジェクト

・任意の配列(空の配列も含む)

・空白のみを含む文字列(“ ”)など

・文字列”false”

偽

・undefined

・null

・false

・0

・NaN

・’’(空文字列)

条件演算子　p96

const doIt = false;

const result = doIt ? "Did it!" : "Didn't do it.";

console.log(result); // Didn't do it.

分割代入(デストラクチャリング)

const obj = { b: 2, c: 3, d: 4 }; // オブジェクトの定義

const {a, b, c} = obj; // 分割代入

console.log(a); // undefined （objにはプロパティaがない）

console.log(b); // 2

console.log(c); // 3

console.log(d); // ReferenceError: d is not defined （エラー。dには代入されていない）

テスト

Q.++xとx++ の違いは？　p86

A. ++x ← 変数の値を変換した「後」の値を結果として返す

x++ ← 変数の値を変換する「前」の値を結果として返す

Q. 厳密等価演算子と等価演算子の違いは？　p88

A. 厳密等価演算子 ← データ型も同じでなければならない

　 等価演算子 ← 同じ値に変換されれば等価

Q. 短絡評価とは何か？　p94

A. 短絡評価 ← 真偽値表から考えると最初のオペランドで評価できてしまう場合、後続のオペランドが評価されなくなること。最初のオペランドのみ参照する

Q. type of 演算子とは？ p99

A. 被演算子のデータ型を表す文字列を返す

**６章　関数**

メモ

・何らかの値の後に()をつけると関数だと判断して呼び出す　p107

・関数の中でオブジェクトのプロパティの値を変更すると、呼び出し側にも影響を与える　＝　プリミティブだと影響を与えない(関数内のみで終了)。つまり、関数の外のプロパティも変更されてしまう

・javascriptのプリミティブは値型であると言われており、渡される時は値がコピーされる。オブジェクトは参照型であり、渡される時は渡す方の変数も渡される方の変数も同じオブジェクトを参照する

・p110のオブジェクト関連のコードを再読

・p114のthis関する説明から最後まで再読

・

アロー関数

・functionという単語を省略できる

・引数が一つならば()を省略できる

・関数本体が一つの式からなる場合、{}とreturn文を省略できる

テスト

Q.メソッドとは何か？

A. オブジェクトのプロパティとして指定される関数のこと

Q.callとapplyとbindの違いは何か？　p119

A.call ← 第一引数に束縛したい値を持ち、残りの引数は呼び出す関数の引数

　apply ← 引数を配列として受け取る

　bind ← thisの値を永続的に結びつけることができる

**７章　スコープ**

・変数の隠蔽　理解できなかったので再読

{ /\* 外側のブロックの始まり \*/

let x = { color: "青" };

let y = x; /\* yとxが同じオブジェクトを参照する \*/

let z = 3;

{ /\* 内側のブロックの始まり \*/

let x = 5; /\* 外側のxが「マスク」される（隠されてしまう） \*/

console.log(x); // 5

console.log(y.color); // 青 （yによって参照されているオブジェクトは相変わらずスコープに入っている。外側のブロックのxもスコープには入っている）

y.color = "赤";

console.log(z); // 3 （zはマスクされていない）

} /\* 内側のブロックの終わり \*/

console.log(x.color); // 赤 （内側のスコープでオブジェクトに変更がなされた）

console.log(y.color); // 赤 （xとyは同じオブジェクトを参照している）

console.log(z); // 3

} /\* 外側のブロックの終わり \*/

テスト

Q. スコープとは何か？

A. 変数、定数および引数が有効な範囲(いつ、どこで定義されていると見なされているのか、どこで値を参照したり変えたりできるのか)を決めるもの

Q. ブロックスコープとは何か？

A. ブロックスコープで定義された識別子はそれを囲むブロックでしか有効にはならない

Q. クロージャとは何か？

A. 関数の周囲にスコープを閉じ込むもの

Q. constとvarとletの違いは何か？

A.varを使った変数には巻き上げという作用が生じ、宣言の前でも参照することができる。ただし、値の参照はできない

**８章　配列**

p146の後半〜p156まで理解できなかったので、再読すること

メモ

メソッド関連の部分だけ復習する

// 配列の長さを大きくする

arr1[4] = 5;

console.log(arr1); // [1, 2, 3, , 5]

console.log(arr1[3]); // undefined

console.log(arr1.length); // 5

// 配列の最後の要素より後ろの添字を使う（代入ではない）->大きさは変わらない

console.log(arr2[10]); // undefined

console.log(arr2.length); // 3

// Arrayコンストラクタ（通常は使われない）

const arr5 = new Array(); // 空の配列を生成

console.log(arr5); // []

const arr6 = new Array(1, 2, 3);

console.log(arr6); // [ 1, 2, 3 ]

const arr7 = new Array(2); // 長さ2の配列（各要素はundefined）

console.log(arr7); // [ , ]

console.log(arr7[1]); // undefined

const arr8 = new Array("2");

console.log(arr8); // ['2']

配列要素を操作するメソッドには、内容を変更するものと新しい配列を返すものがある。例えば、pushは配列を変更するが、concatやsliceやspliceは新しい配列を返す。

メソッド concat

複数の配列を最後に追加する

let arr = [1, 2, 3]

let arr2 = arr.concat(4, 5, 6)

console.log(arr) // [1, 2, 3]

console.log(arr2) // [1, 2, 3, 4, 5, 6]

メソッド slice

第一引数に指定した場所から、第二引数に指定した場所の前までの要素からなる部分配列を作成する。第二引数を省略すると最後まで切り取る

let arr = [11, 12, 13, 14, 15]

arr2 = arr.slice(2, 4)

console.log(arr2) // [13, 14]

テスト

Q. push, pop, shift, unshiftの違いは？

A. 全て、配列そのものを変更する

push ← 配列の最後に要素を追加する。返り値は配列の長さ

　 pop ← 最後の要素を削除する。返り値は削除した値

　 shift ← 先頭の要素を削除する。返り値は削除した値

　 unshift ← 先頭に要素を追加する。返り値は配列の長さ

**9章　オブジェクトとオブジェクト指向プログラミング**

新しいインスタンスが生成されるときにはコンストラクタが呼び出される。

コンストラクタがオブジェクトのインスタンスを初期化する。

クラスにおけるthisはメソッドが結び付けられる具体的なインスタンス。

index Of ← 指定した値に厳密に等しい値の配列の添字を返す。存在しなかったら-1を返す。

プロトタイプ

あるクラスのインスタンスに対して使えるメソッドを参照する時に、プロトタイプメソッドを参照している。

動的ディスパッチ(ディスパッチは呼び出しという意味)

オブジェクトのメソッドにアクセスしようとしてそれが存在していないと、そのオブジェクトのプロトタイプを参照する。

プロトタイプチェイン

オブジェクトのプロトタイプが見つからない場合、プロトタイプのプロトタイプを参照する。これらが繰り返されるため、プロトタイプチェインと呼ばれる。

静的メソッド

インスタンスを生成しなくても、オブジェクトから直接呼び出せるメソッド。

静的メソッド使う理由

静的プロパティ/静的メソッドは機能的にはグローバルプロパティ/グローバルメソッドとは何ら変わりません。ではなぜ静的プロパティ/静的メソッドを使うのかというと、「名前の競合」を回避するためです。  
クラスごとのプロパティ/メソッドである静的プロパティ/静的メソッドを使えば、たくさんのライブラリを使う開発であっても機能ごとにグローバルオブジェクトのように使えるので、大変勝手が良い。  
よって、グローバル変数/関数は出来るだけ減らし、静的プロパティ/静的メソッドに情報をまとめるのが好ましい。

super()

スーパークラスのコンストラクタを呼び出す関数。サブクラス側ではこの関数を必ず呼び出す必要がある。

ポリモーフィズムとは「あるインスタンスをそのインスタンスが属するクラスのメンバーとして扱うだけでなく、スーパークラスのメンバーとしても扱う」ことを意味する

ミックスイン

JavaScriptは多重継承ができないため、それらを解消するためにプロトタイプを使って擬似的に多重継承をすること

**10章　マップとセット**

イテレーション

反復処理のことであり、例えばforループなどがある

ガーベッジコレクション

不要になったメモリ機能を自動的に開放する機能

Map

同じデータ構造であるオブジェクトとの違い

・プロトタイプがないため、意図しないマッピング(キーと値の対応)が生じない

・キーと値の組みがいくつあるのか分かる(プロパティsize)

・キーが文字列あるいはシンボルに限られない。つまり、オブジェクトをキーとして値と対応づけることができる

WeakMap

・キーがオブジェクトでなければならない

・キーがガーベッジコレクションの対象になる

・イテレーションしたりクリアできない

IIFE

定義した瞬間に即時実行される関数

Set

・重複の許されないデータ集合(構造)

WeakSet

・オブジェクトだけを含むことができるセット

・イテレーションできない

11章　例外とエラー処理

Errorオブジェクト

JavaScriptには組み込みのErrorオブジェクトがあり、インスタンス生成時にエラーメッセージを指定することができる

例外処理(try catch)

何かtryして、それが例外を引き起こした時にcatchによってその例外がキャッチされる。エラーが起こると制御はtry文に移る

例外のスロー

例外を明示的にスローできる

finally

tryブロックに書いたリソースに関する処理を確実に解放する

12章 イテレータとジェネレータ

イテレータ ← 繰り返しのための機構。

イテレータオブジェクト ← メソッドnextを呼び出すことで次々と要素を取り出せるオブジェクト

配列はイテレーション可能(for of)なオブジェクトであるが、イテレータオブジェクトではない。つまり、メソッドnextが使えない。メソッドnextを使うにはvaluesというメソッドでイテレータオブジェクトに変換する。

イテレーション可能なオブジェクト(for ofで各要素にアクセスできる)

配列、文字列、マップ、セット

Symbol.iterator (イテレータプロトコルを実装する)

イテレータの振る舞いをするオブジェクトを戻すもの。配列などはイテレーション可能なオブジェクトであるから、メソッド[Symbol.iterator]を持っており、そのメソッドを利用すればイテレータオブジェクトを取得することができる。

ジェネレータ (←関数の一種)

・ジェネレータを呼び出すとすぐには実行されず、まずはイテレータが戻される。その後で、イテレータのメソッドnextを呼び出す度に実行が進む。

・funcion\* と定義し、値を返すためにはyieldを使う。

・yieldは式なので、評価の結果、何らかの値になる。ここではnext呼び出し時の引数の値になる。

・ジェネレータのどこかでreturnを呼び出しするとdoneがtrueになり、valueプロパティはreturnに指定した値になる

13章 関数のさまざまな利用方法

サブルーチンとしての関数

繰り返しを軽減させる仕組み

純粋関数

・同じ入力に対して同じ出力を返す

・副作用を持たない。つまり、プログラムの状態を変えない

・p218の後半からのコードが分からない

変数を使えるところならば関数も使える

・関数を参照する変数でエイリアス(別名)を作る。

・関数を配列に保存する。

・関数を別の関数に引数として渡す

・関数から戻り値として関数を返す

14章 非同期プログラミング

コールバック ← 将来のある時点で呼び出される関数

コールバック関数 ← 関数に引数として渡されて、関数内から呼び出される関数

疑問

・p233のコールバックと非同期のコードに関して

・全体的に再読すること

15章 日時

・日時は内部的にはUNIXエポック(UTC1970年一月一日)からのミリ数秒で表される。

・日時を生成する際にはタイムゾーンに注意が必要。

・日時の表示を細かく指定したいのならばMoment.jsが検討に値する。