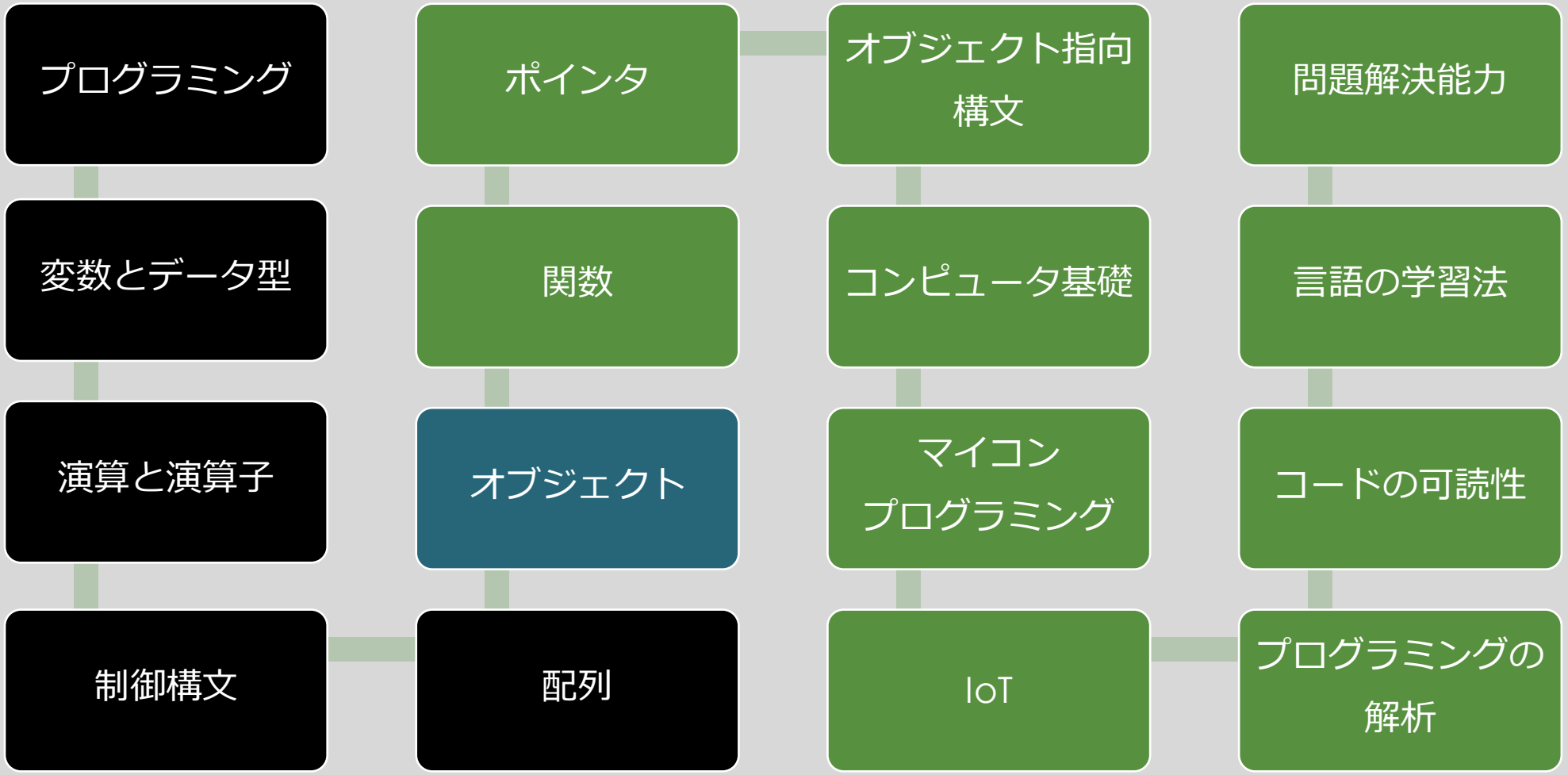
多言語プログラミング

～プログラミングにおける骨格～

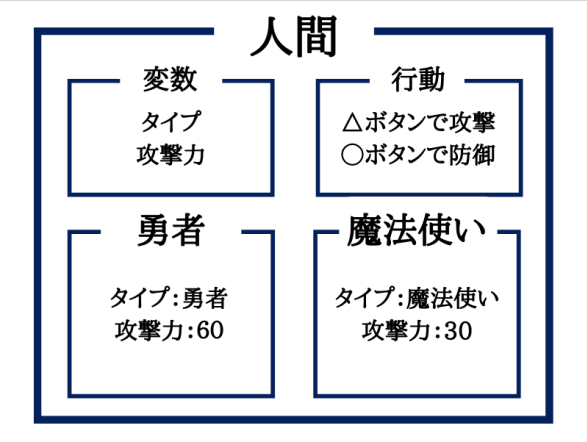
オブジェクト

～より複雑なデータをより扱いやすく～

進度


プログラミング
変数とデータ型
演算と演算子
制御構文
ポインタ
関数
オブジェクト
配列
オブジェクト指向構文
コンピュータ基礎
マイコンプログラミング
IoT
問題解決能力
言語の学習法
コードの可読性
プログラミングの解析

オブジェクトとは



- 配列

  要素番号(index)を指定してアクセスする単一属性のデータ群

- 連想配列

  設定した名前(key)を指定してアクセスする単一属性のデータ群

- オブジェクト

  複数属性で構成され、

  それぞれの名前を指定してアクセスするひとつのモノ

連想配列

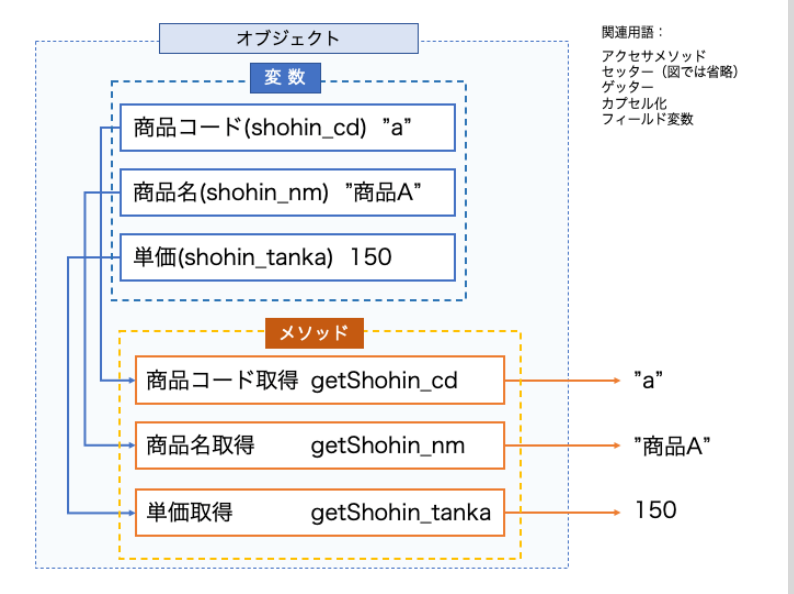
```
let student =
        {name: '太郎', birthday: '2002/02/02'};
console.log(student.name);
//太郎が表示される
console.log(student['birthday']);
//2002/02/02が表示される
```

◦ 名前をキーにアクセスできる配列

let 変数名 ＝ { キー名:値, キー名:値,...}
として初期化。キーは文字列でも数値でも可

変数名.キー名（ドット演算子）
変数名['キー名']（添字演算子）
としてアクセスが可能。ドット演算子については後述。

オブジェクトの構成



- プロパティ
  - データプロパティ
    オブジェクトの状態や属性を表す情報
  - アクセサープロパティ
    データプロパティへ外部変数からの入出力を行う
    ゲッター関数/セッター関数
- メソッド

  オブジェクトを操作するツールであり、つまり関数のこと

  関数の章で詳述

  プロパティとメソッドを合わせてオブジェクトのメンバという

オブジェクトの基本要素

- インスタンス化

  オブジェクトのひな型からコピーを生成すること

- インスタンス

  インスタンス化によって生成されたコピー

- コンストラクタ

  オブジェクト内に用意されているオブジェクトと同名のメソッド

インスタンスと初期化

```
let date = new Date( '2003/03/14 00:00' );
//Dateオブジェクトのインスタンス化
```

let 変数名 ＝ new オブジェクト名([引数,…])

としてオブジェクトをインスタンス化して、変数を宣言する

new修飾子：オブジェクトをインスタンスするときに使用

- 変数のデータ型はインスタンス化したオブジェクト型となる

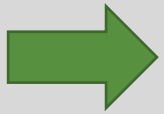
- 引数は先述のコンストラクタに与えられ、オブジェクトの初期化を行う

プロパティやメソッドの参照

```
console.log(date.getFullYear());
//dateのgutFullYearメソッドにより
//西暦年を4桁で取得
```

変数名.プロパティ名[= 設定値];

としてインスタンス変数のデータプロパティにアクセス



ゲッターとセッターを呼び出してアクセスしている

変数名.メソッド名([引数,...]);

としてインスタンス変数のメソッドにアクセス

- .（ドット演算子）

インスタンス変数のプロパティやメソッドにアクセスするための演算子

静的プロパティ/静的メソッド

```
let now = Date.now();
//UNIXエポックからの経過ミリ秒を取得
```

- 静的プロパティ/メソッド

  インスタンス化をせずにオブジェクトそのものにアクセスが可能

オブジェクト名.プロパティ名[= 設定値];
オブジェクト名.メソッド名([引数,...]);

- 静的メソッドやプロパティを使用することで、

  グローバル変数を減らすことが出来る

組み込みオブジェクト

- 基本的なオブジェクト、JavaScriptに組み込まれている
- 特別な宣言や定義なしに利用可能
- インスタンス化やnew修飾子等も基本的には使用しない

```
let str = '電研';
//Stringオブジェクトとなる

let num = 5;
//Numberオブジェクトとなる
```

演習６“オブジェクト”

1. Stringオブジェクトを使用し、
   ”電気技術研究会”の文字数を数える
2. Numberオブジェクトを使用し、
   円周率を有効数字6桁で出力する
3. Dateオブジェクトを使用し、
   “入学した年/4/1 0:00”までの経過ミリ秒を取得する

演習６“オブジェクト” ヒント

1. 変数名.lengthを用いる
2. 円周率はMath.PI、桁決めは変数名.toPrecision(桁数)
3. Dateオブジェクトは変数宣言時に日時指定
4. ↑＋変数名.getTime()で経過ミリ秒を取得

```
let date = new Date( '2003/03/14 00:00' );
//Dateオブジェクトのインスタンス化
```

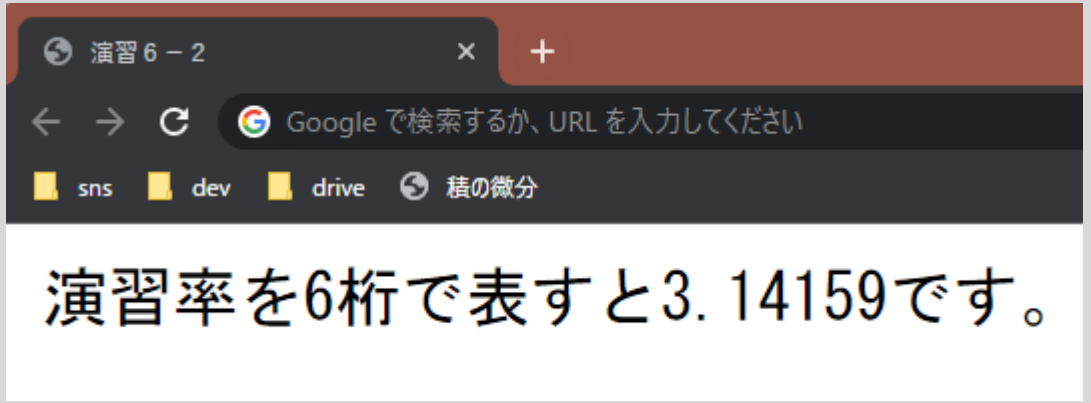
```
console.log(date.getFullYear());
//dateのgutFullYearメソッドにより
//西暦年を4桁で取得
```

演習６“オブジェクト”①　回答

```
let str = "電気技術研究会";
let len = str.length;
document.writeln(`${str}の文字数は${len}です。`);
```

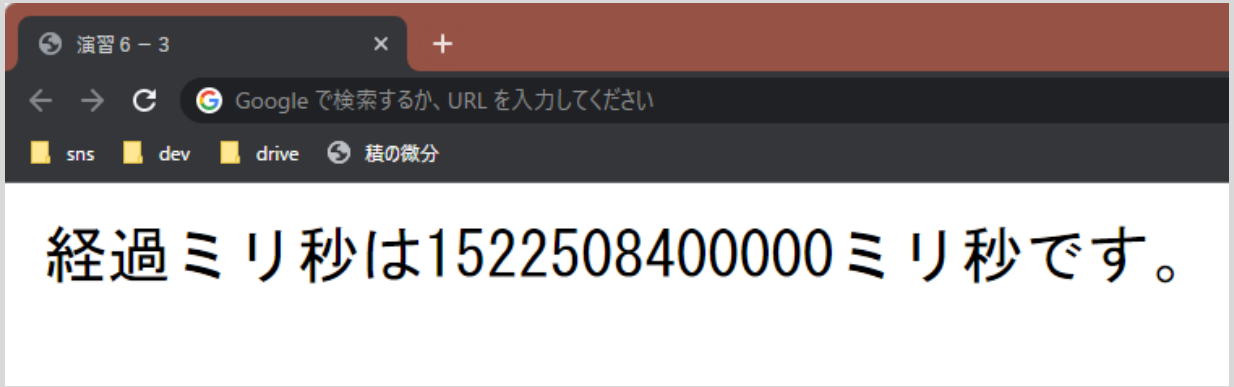
演習６“オブジェクト”②　回答

```
let pi = Math.PI;
let pi6 = pi.toPrecision(6);
document.writeln(`演習率を6桁で表すと${pi6}です。`);
```



演習６“オブジェクト”③　回答

```
let date = new Date('2018/04/01 00:00');
let ms = date.getTime();
document.writeln(`経過ミリ秒は${ms}ミリ秒です。`);
```



オブジェクトまとめ

- 連想配列は名前をキーにしてアクセスが出来る配列
- オブジェクトは複数属性で構成される、ひとつのモノ
- オブジェクトにはプロパティとメソッドがある
- オブジェクトはインスタンス化して使用するが、
  静的プロパティ/メソッドはインスタンス化しなくても使用可能

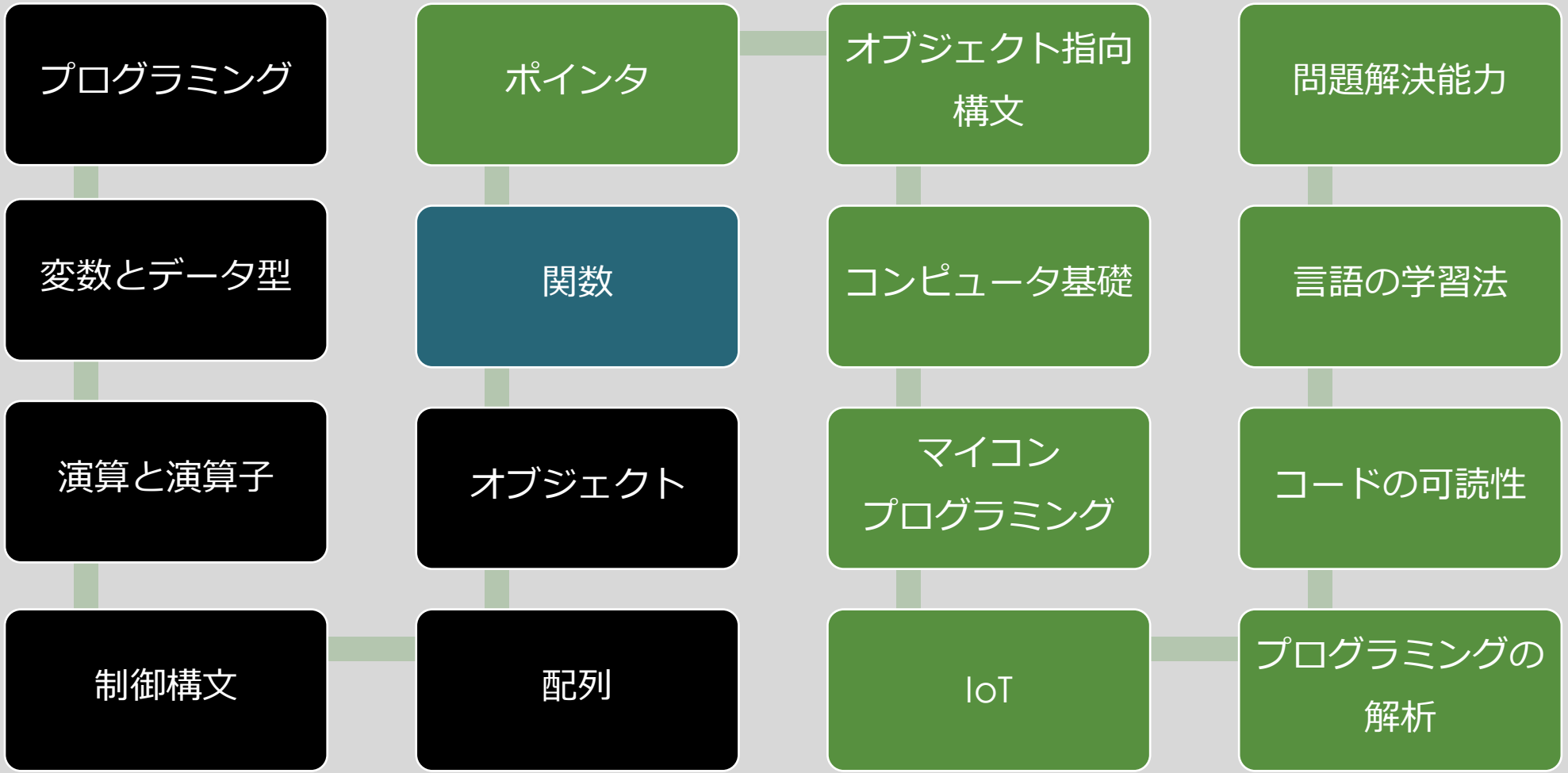
参考文献

- https://products.sint.co.jp/topsic/blog/object-oriented
- https://blog.codecamp.jp/object-orientation-explanation

関数

～繰り返し処理をひとつに～

進度


プログラミング
変数とデータ型
演算と演算子
制御構文
ポインタ
関数
オブジェクト
配列
オブジェクト指向構文
コンピュータ基礎
マイコンプログラミング
IoT
問題解決能力
言語の学習法
コードの可読性
プログラミングの解析

関数

- 処理をひとつにまとめて名前を付けたもの
- 関数はプログラム中のどこでも呼び出すことが可能
- メインループもサブルーチンも関数であり、
  プログラムは全て関数でできている

基本要素

- 定義
  関数の宣言を行う。関数の処理などを決定する
- 呼び出し
  関数の処理を実際に行う
- 実引数
  呼び出し元が関数に与える値。関数側では仮引数という
- 戻り値
  関数の呼び出し後、関数から戻ってくる値

関数の定義

```
int sum(int a, int b){
    int result = a + b;
    return result;
}
```

- 以下のように宣言する（静的はC、動的はJSの例）
  - 戻り値の型 関数名(仮引数の宣言) { 処理}
  - function 関数名(仮引数){処理}

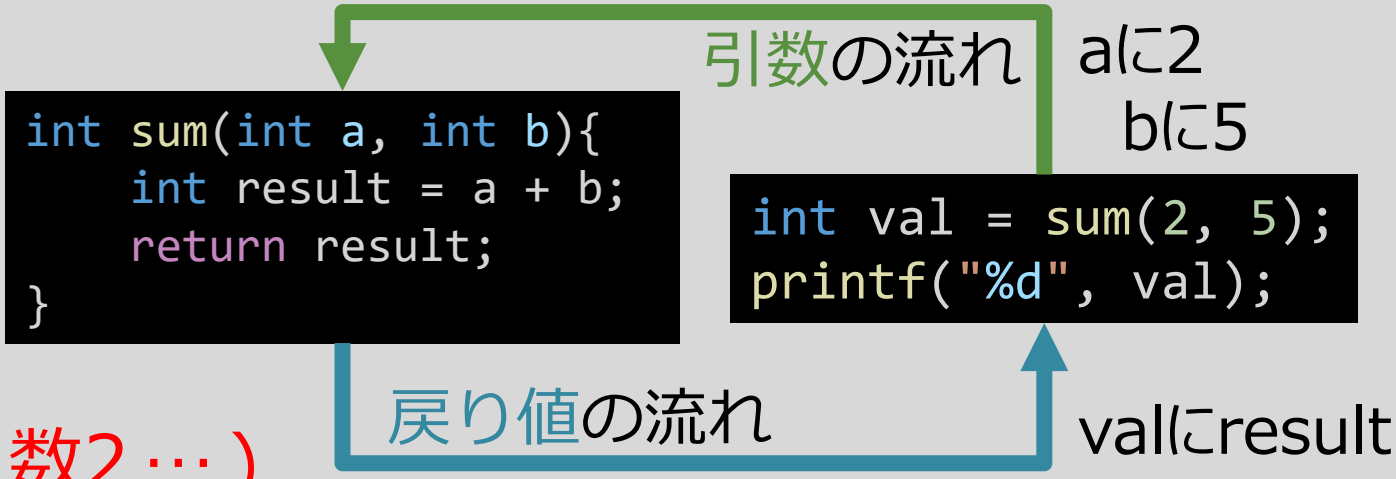
```
function sum(a, b){
    return a + b;
}
```

- return命令で呼び出し元に返す戻り値を指定し終了できる
- function命令では引数にletなどを使用して宣言はしない
- 戻り値や引数を指定しない場合、void型を使用する

オブジェクト指向言語における関数

- オブジェクトが持つ固有の関数：メソッド
- 前頁のように宣言した関数はユーザ定義関数となる
- 両者に機能的な違いはない
- JavaScriptにてユーザ定義関数の宣言法は他にも
  Functionオブジェクト、関数リテラル、アロー関数などがある

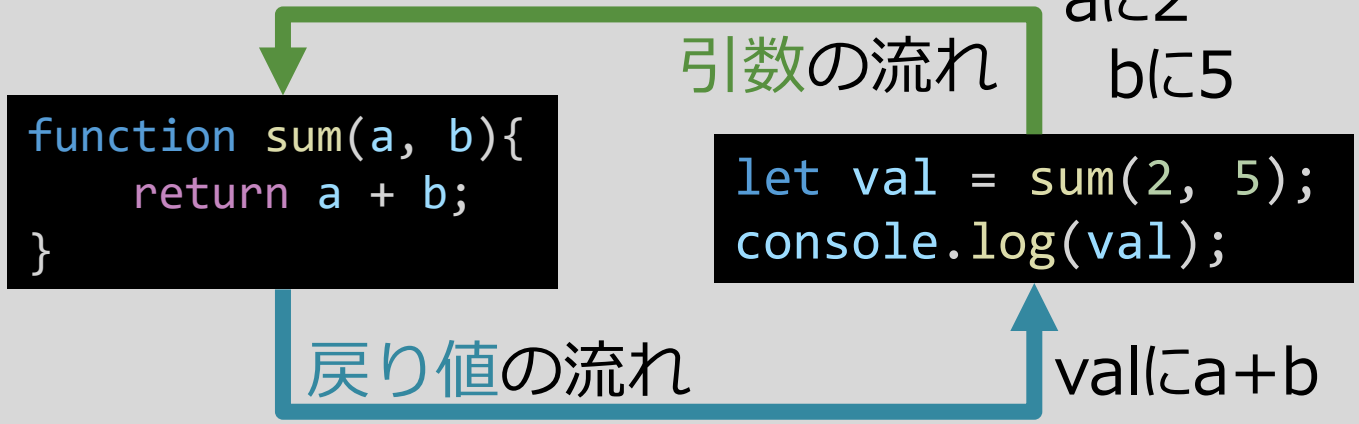
関数の呼び出し



- 関数名(実引数1，実引数2…)

  として関数の処理を実行し値を取りだすことが可能

- 実引数を与えて処理を実行し
  return文で指定した値が返却される



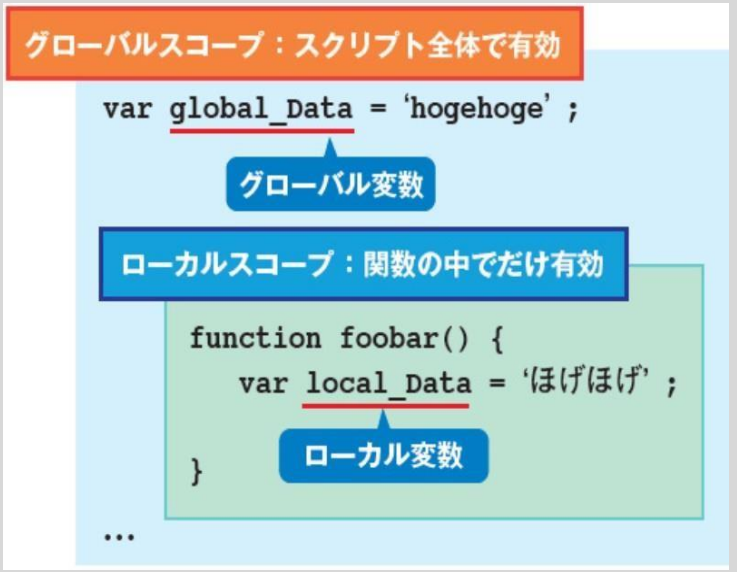
関数のオーバーロード

- C＋＋などでは同じ名前で処理の違う関数を複数個宣言できる
- 引数の個数やデータ型によって振り分けが行われる
- 戻り値のみ異なる関数はエラーになる

```
int val1 = sum(2, 5);
int val2 = sum(2, 4, 5);
```

```
int sum(int a, int b){
    int result = a + b;
    return result;
}
int sum(int a, int b, int c){
    int result = a + b + c;
    return result;
}
```

変数の有効範囲



- ブロック有効範囲

  関数のブロック({ から} まで) での有効範囲

  宣言した変数(ローカル変数) は外から扱えない

- ファイル有効範囲

  どのブロックにも属さない一番外側での有効範囲

  宣言した変数(グローバル変数) はファイル内のどこからでも扱える

  ➡ 変更されるタイミングが不明なので可読性は×

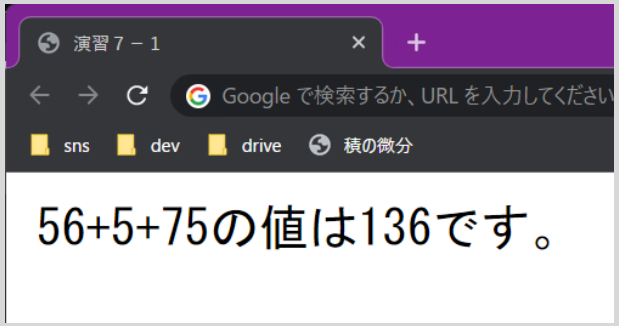
ヘッダファイルとライブラリ

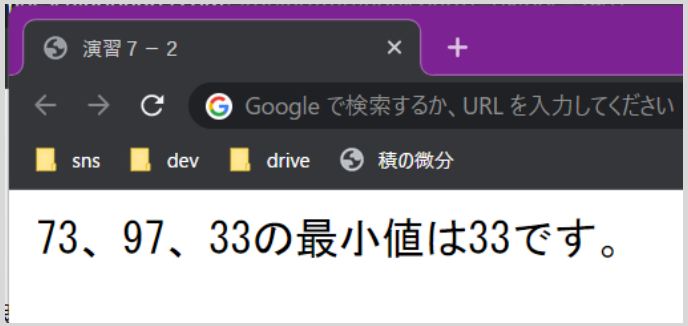
- ～.h(ヘッダファイル)では、
  ライブラリにて使用される関数の宣言をしている
- #include命令を使用することでライブラリ関数が使用可能に
- Arduinoではセンサーなどを扱うときに使用されることがしばしば
- JavaScriptやC#など多くのオブジェクト指向言語では
  ヘッダファイルではなくライブラリに様々なメソッドがまとめられている

演習７“関数”

1．与えられた3つの乱数の和を返す関数を作成

2．与えられた3つの乱数の最小値を返す関数を作成





演習７“関数” ヒント

```
if(条件式){
    //条件式がtrueの場合
}else{
    //条件式がfalseの場合
}
```

1. CであればSerial.println()、
   JavaScriptであればdocument.writeln()
2. 必要に応じて変数を自分で作成しよう
3. ②はif文のネストとかを使えばできる…？

```
randomSeed(analogRead(0));
//毎回ランダムにするために必要
int min = 1;
int max = 5;
int rand = random(min, max);
```

```
let max = 5, min = 1;
let rand1 =
    Math.floor(Math.random() * (max - min) + min);
```

```
function sum(a, b){
    return a + b;
}
```

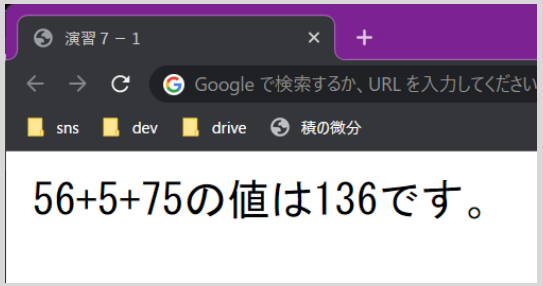
```
int sum(int a, int b){
    int result = a + b;
    return result;
}
```

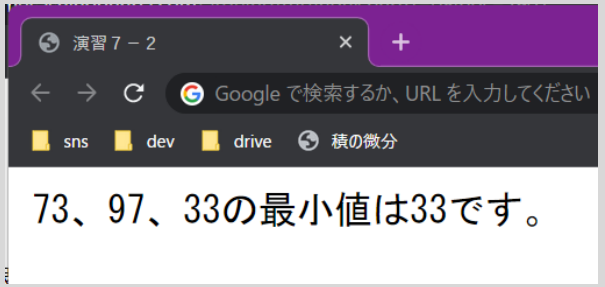
演習７“関数” 回答（JavaScript版）

```
function sum(a, b, c){
    let result = a + b + c;
    return result;
}

let a = Math.floor(Math.random() * 100);
let b = Math.floor(Math.random() * 100);
let c = Math.floor(Math.random() * 100);

document.writeln(`${a}+${b}+${c}の値は
${sum(a,b,c)}です。`);
```






```
function min(a, b, c){
    let min = a;
    if(a < b){
        if(a > c){
            min = c;
        }
    }else if(b < c){
        min = b;
    }else{
        min = c;
    }
    return min;
}

let a = Math.floor(Math.random() * 100);
let b = Math.floor(Math.random() * 100);
let c = Math.floor(Math.random() * 100);

document.writeln(`${a}、${b}、${c}の最小値は
${min(a,b,c)}です。`);
```

演習７“関数” 回答（Arduino版）

```
int sum(int a, int b, int c){
    return a + b + c;
}
```

```
    Serial.begin(9600);
    randomSeed(analogRead(0));

    int a = random(100);
    int b = random(100);
    int c = random(100);

    Serial.print(a);
    Serial.print("+");
    Serial.print(b);
    Serial.print("+");
    Serial.print(c);
    Serial.print("=");
    Serial.println(sum(a, b, c));
```

```
87+72+17=176
47+44+87=178
65+0+57=122
48+69+22=139
```

```
int minimum(int a, int b, int c){
    int min = a;
    if(a < b){
        if(a > c){
            min = c;
        }
    }else if(b < c){
        min = b;
    }else{
        min = c;
    }
    return min;
}
```

```
a : 99, b : 73, c : 46, min : 46
a : 85, b : 22, c : 94, min : 22
a : 13, b : 71, c : 45, min : 13
```

```
randomSeed(analogRead(0));

int a = random(100);
int b = random(100);
int c = random(100);

Serial.print("a : ");
Serial.print(a);
Serial.print(", b : ");
Serial.print(b);
Serial.print(", c : ");
Serial.print(c);
Serial.print(", min : ");
Serial.println(minimum(a, b, c));
```

関数のまとめ

- 関数の操作には定義と呼び出しがある
- 関数内では仮引数として受け取り戻り値をreturnする
- 呼び出し元では実引数を与えて戻り値を受け取る
- 変数の有効範囲は関数のブロックにより決定する