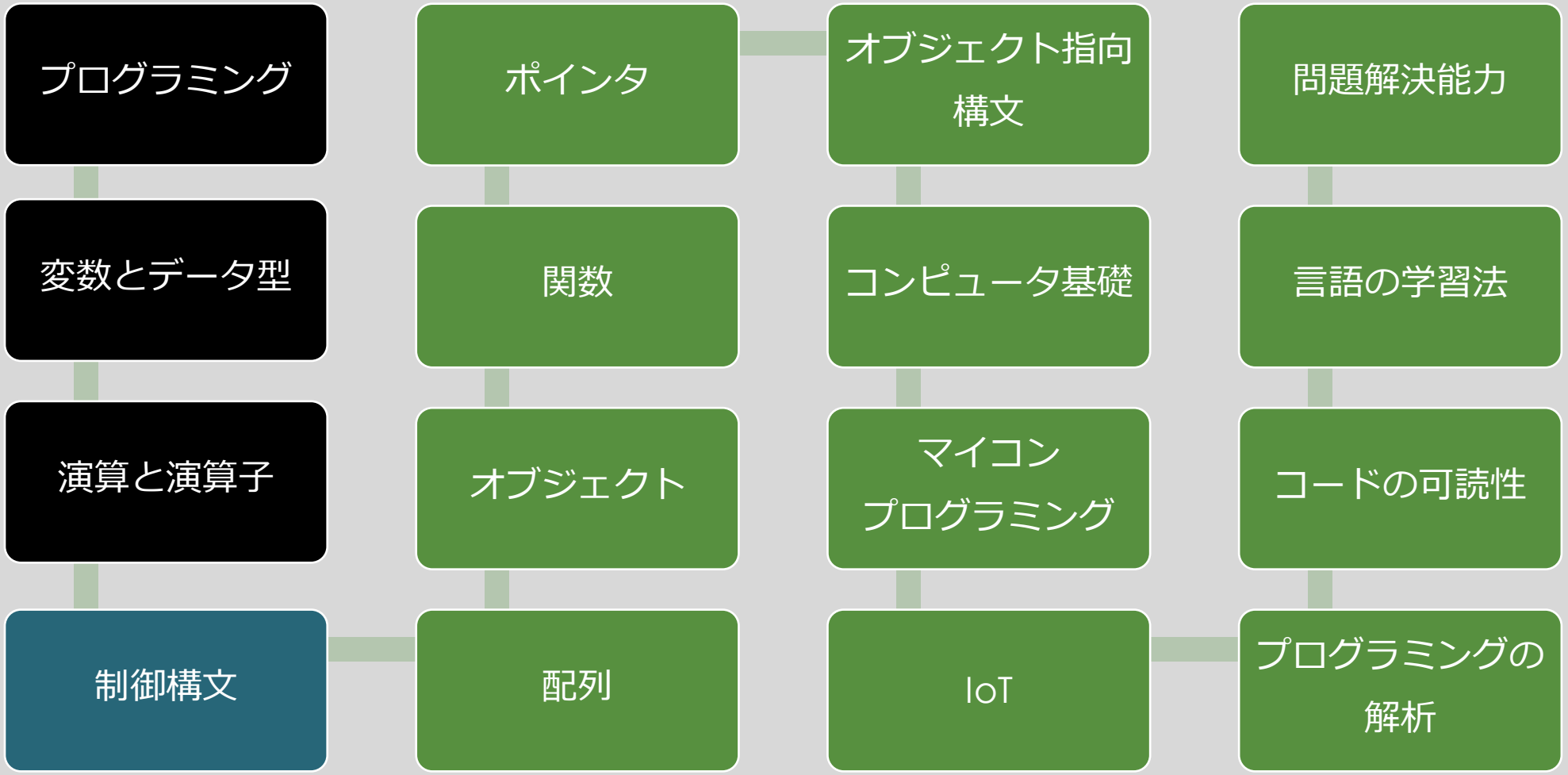
多言語プログラミング

～プログラミングにおける骨格～

制御構文

～条件分岐と繰り返し～

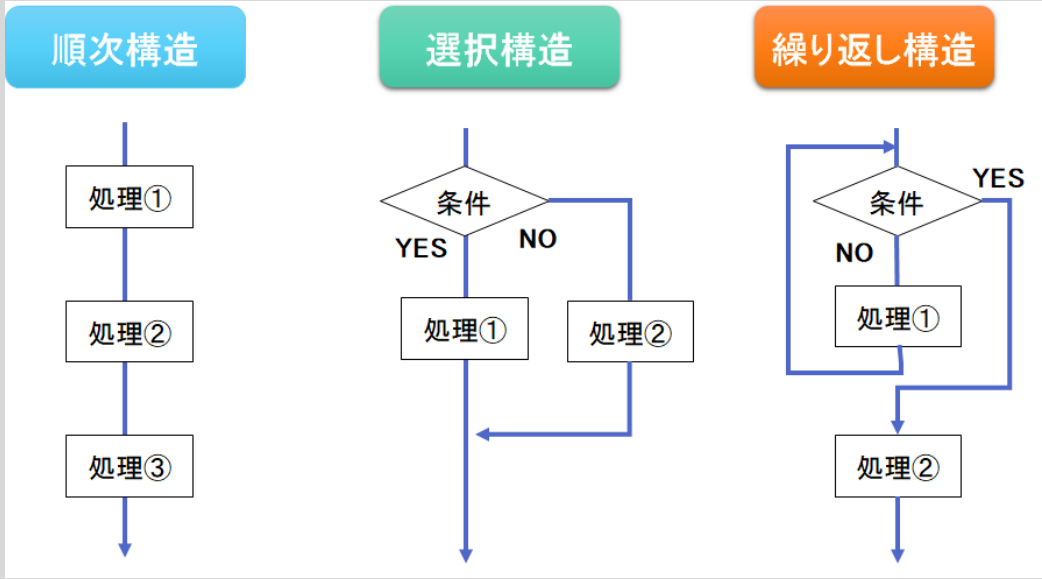
進度


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

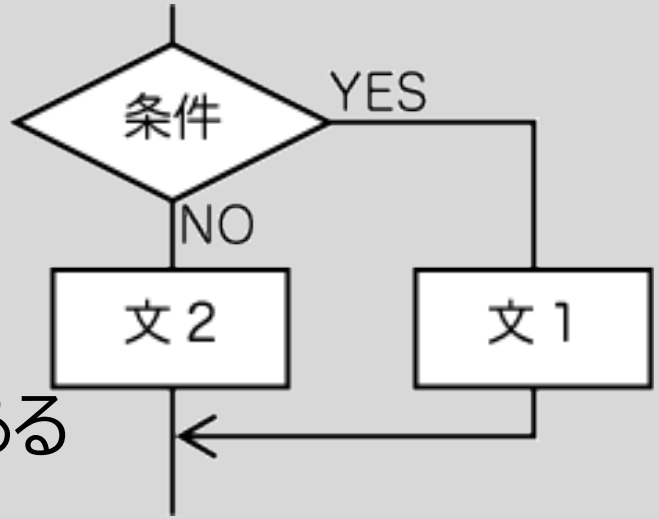
制御構文

構造化プログラミングの手法

1. 記述された順番に処理を行う順次
2. 条件によって処理を分岐する選択
3. 特定の処理を繰り返し実行する反復



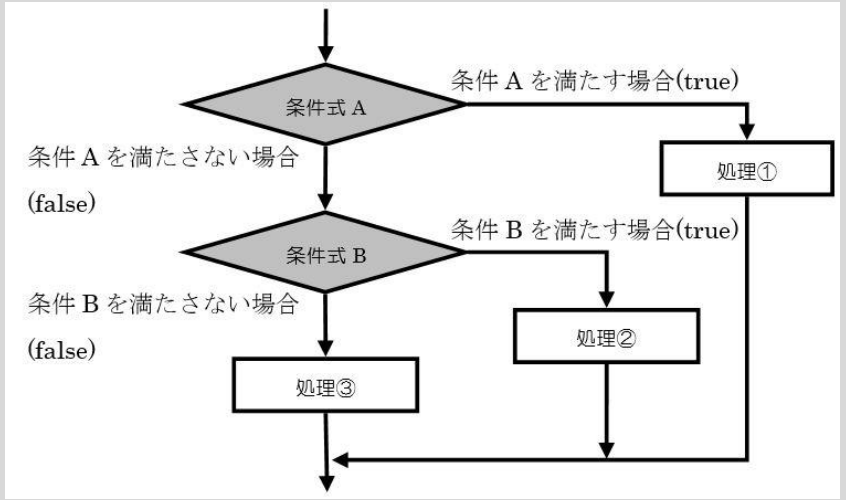
if文～条件による処理の分岐～



- 時と場合に応じて、処理の分岐が必要
  →if文やswitch文がある
- if構文は真のとき実行するifと、
  偽のとき実行するelseによって成立
- 条件式は比較演算子と論理演算子
  によって立てる

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

if文～条件による処理の分岐～



else-ifによる多岐分岐

- else if文を用いて多岐の分岐が可能

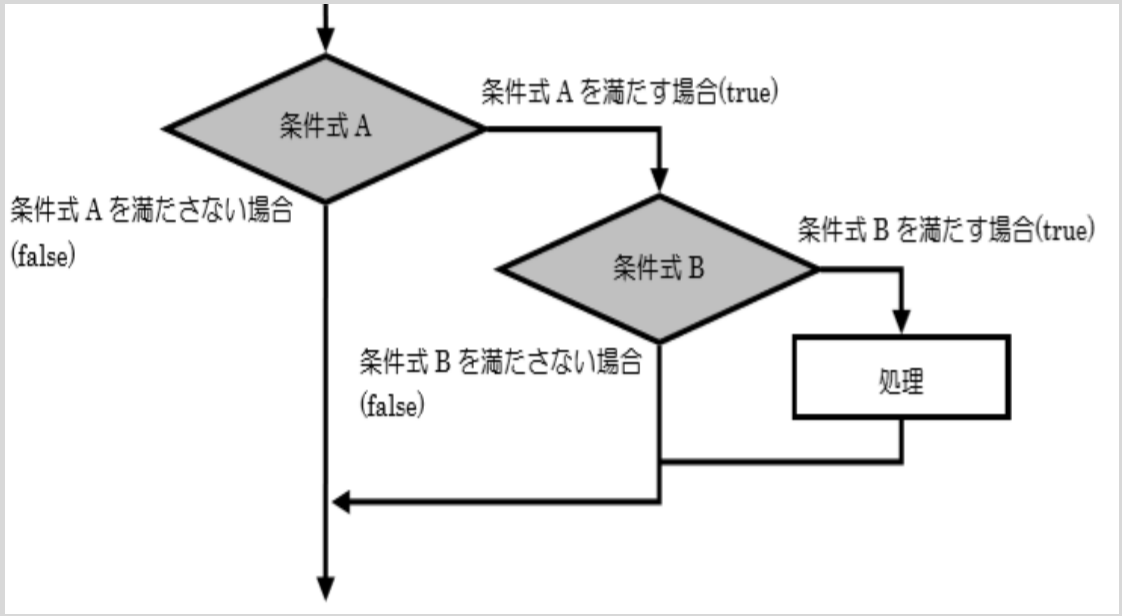
- 上から順に実行するため、評価は
  条件式1→条件式2の順番になる

※switch文でも多岐分岐が可能
  可読性のために使い分けが必要

```
if(条件式１){
    //条件式1がtrueの場合
}else if(条件式２){
    //条件式1がfalseで
    //条件式2がtrueの場合
}else{
    //条件式が全てfalseの場合
}
```

if文～条件による処理の分岐～

if文による入れ子構造（ネスト）



◦if文を入れ子にすると、
より複雑な分岐が可能に
→ネストともいう

※ほかの制御構文でもネストは可能だが、可読性も考慮するべき…
→”コードの可読性”の章で詳述

switch文

- 変数の値によって多岐分岐できる便利な構文switch文
  →同値演算子(＝＝)による多岐分岐
- 以下の手順で処理が分岐
  1. 先頭の式を評価
  2. 上から同値演算を行い、一致する
     case句を実行
  3. 2の手順で見つからない場合、
     default句を実行する

```
switch(式){
    case 値1:
        //式が値1の場合
        break;
    case 値2:
        //式が値2の場合
        break;
    default:
        //式が当てはまらない場合
        break;
}
```

switch文

break文の重要性

- break文は処理の終わりを表す
  →breakがないと、下のcase文まで処理が続行する
- フォールスルー
  break文をわざと省略し、次のbreakまで処理を貫通させる

```
let rank = 'B’
let result;
//フォールスルーの例
//ランクによって場合分け
switch(rank){
    case 'A':
    case 'B':
        result = 'success';
        break;
    case 'C':
        result = 'false';
        break;
    default:
        result = '';
        break;
}
```

演習４“制御構文”①

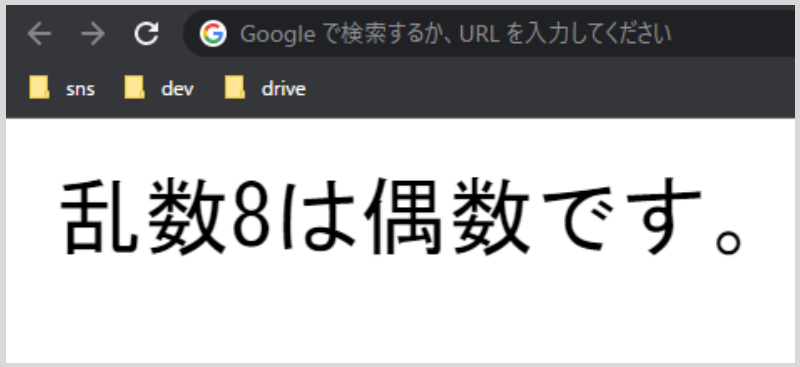
- 1~12までの乱数を発生させ、if文で
  発生させた乱数が奇数か偶数か判断するプログラムを作成する

- 1～12までの乱数を発生させ、switch文で
  発生させた乱数の月の季節を表示するプログラムを作成する

```
let max = 12, min = 1;
let rand = Math.floor(Math.random() * (max - min) + min);
```

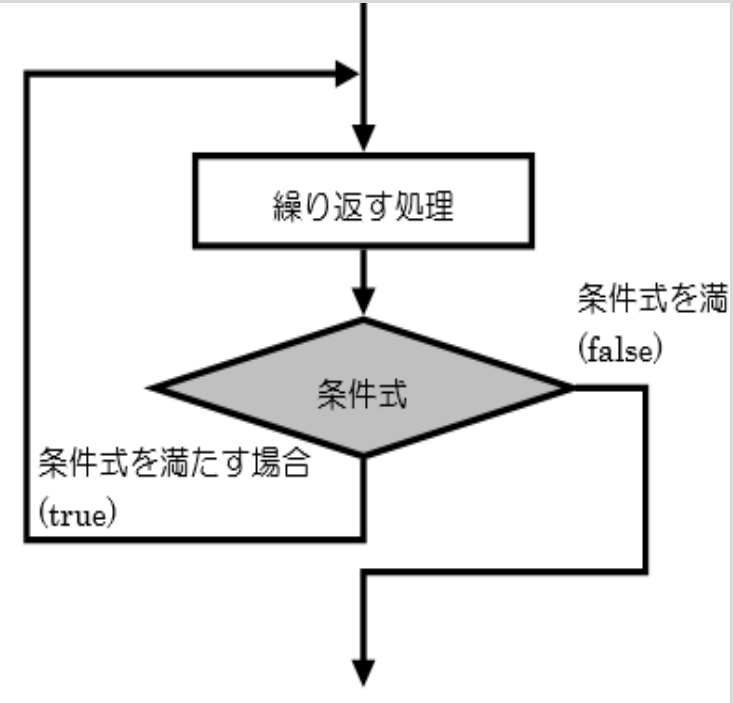
演習４“制御構文”①　回答


```
let max = 12, min = 1;
let rand =
    Math.floor(Math.random()*(max-min)+min);

if(rand % 2 == 0){
    document.writeln(`乱数${rand}は偶数です。`);
}else{
    document.writeln(`乱数${rand}は奇数です。`);
}
```




```
let max = 12, min = 1;
let rand = Math.floor(Math.random() * (max - min) + min);

switch(rand){
    case 11:
    case 12:
    case 1:
        document.writeln(`${rand}月は冬です。`);
        break;
    case 2:
    case 3:
    case 4:
        document.writeln(`${rand}月は春です。`);
        break;
    case 5:
    case 6:
    case 7:
        document.writeln(`${rand}月は夏です。`);
        break;
    case 8:
    case 9:
    case 10:
        document.writeln(`${rand}月は秋です。`);
        break;
}
```

while文①　do~while文

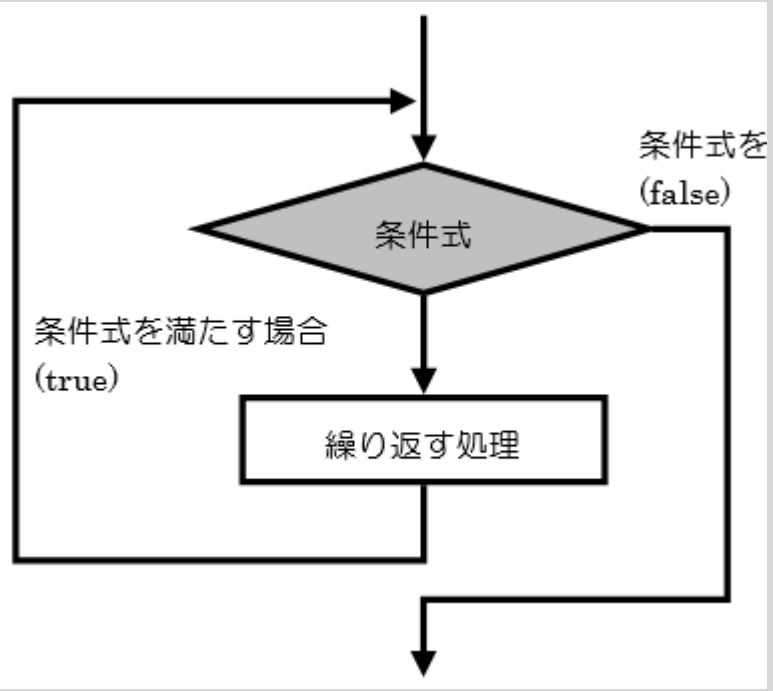


- 処理の分岐と並び、
  処理の反復をする構文も存在
  →while文やfor文

- do~while構文では、do文で処理を実行
  →while文で式を判定し、繰り返し実行

```
do{
//条件式が真の間反復
}while(条件式);
```

while文②　while文



- while構文
  do~whileと同じように
  条件式が真の間処理を繰り返す構文

```
while(条件式){
    //条件式が真の間反復
}
```

- 条件式を意図的に真にすることで
  無限ループができる
  →マイコンの章で詳述

判定の順序

whileとdo~whileでは判定の順序が違う

- 後置判定

  ループの最後に条件式を判定する

  →do~while

- 前置判定

  ループの最初で条件式の判定をする

  →whileやfor

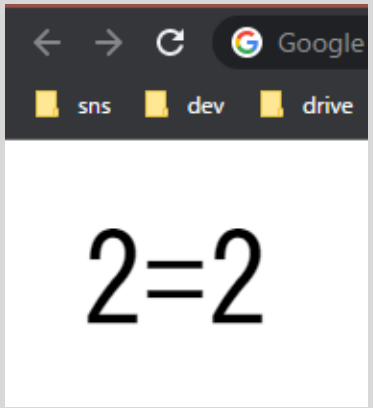
演習４“制御構文”②

1. １~５までの乱数1と乱数2をそれぞれdo文外とdo文内で発生させ、
乱数2が乱数1と同じ値になるまで繰り返すプログラムを作成

2. 上記のプログラムをwhile文で作成し、
do文とwhile文の違いを確認する

演習４“制御構文”②　回答

```
let max = 5, min = 1;
let rand1 = Math.floor(Math.random() * (max - min) + min);
let rand2;
do{
    rand2 = Math.floor(Math.random() * (max - min) + min);
}while(rand2 != rand1);

document.writeln(`${rand1}=${rand2}`);
```




```
let max = 5, min = 1;
let rand1 = Math.floor(Math.random() * (max - min) + min);
let rand2 = 0;
while(rand1 != rand2){
    rand2 = Math.floor(Math.random() * (max - min) + min);
}

document.writeln(`${rand1}=${rand2}`);
```

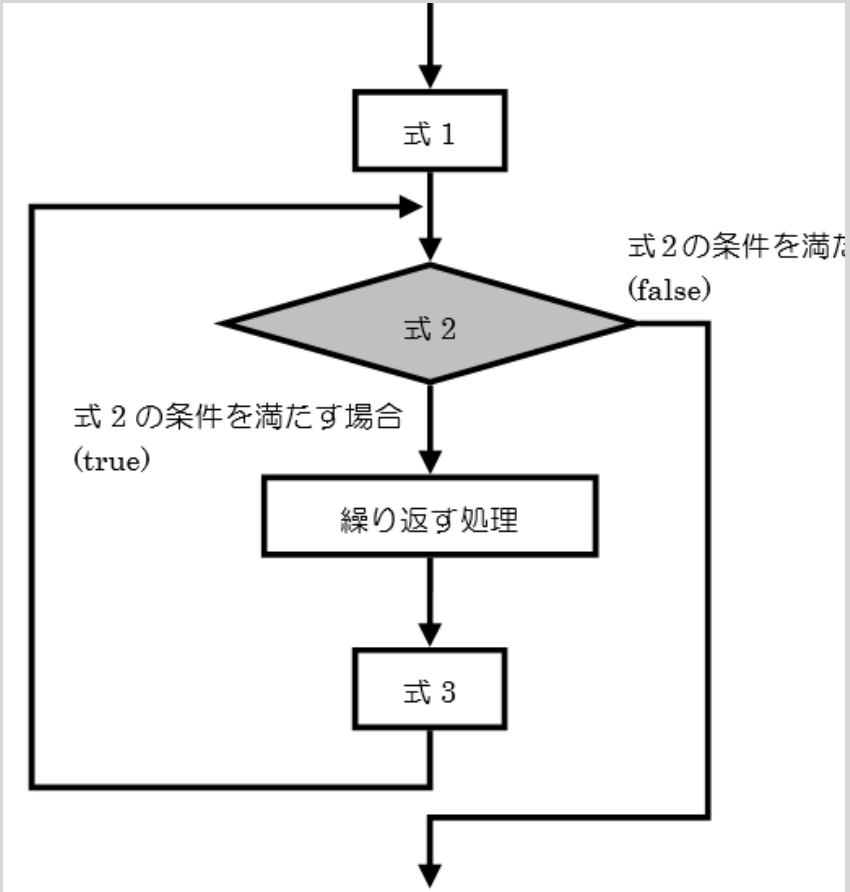
for文

- for構文

  指定された回数処理を繰り返す構文

- 前処理、条件式、後処理の3パートがある

- while文同様無限ループを生成することも可能



for文

```
for(前処理;条件式;後処理){
    //条件式が真の間反復
}
```

- 前処理（省略可）
  ループに入る直前に行う処理、変数の初期化を行うことが多い
- 条件式（継続条件、省略可）
  ループの継続を判定する式。
  省略して無限ループを生成することも可能
- 後処理（省略可）
  ループを終えるたびに行う処理。変数の増減を行うことが多い

無限ループの生成

- 無限ループを用いることで継続的な処理が可能
  →マイコンの章で詳述
- while文での無限ループ
  カッコ内の条件式を真にする

```
while(1){
    //無限に処理が続く
}
```

- for文での無限ループ
  カッコ内の条件式を省略する

```
for(前処理;;後処理){
    //無限に処理が続く
}
```

break文とcontinue文

```
for(let i = 0; i < 5; i++){
        if(i > 3){
                break;
        }
}
```

- break文

  forループを強制的に脱出することができる

  if文と組み合わせて条件で抜けることが多い

```
for(let i = 0; i < 5; i++){
    if(i < 2){
        continue;
    }
}
```

- continue文

  forループを1度スキップして次のループに移ることができる

  breakはfor文すべての処理をスキップするのに対し
  continueは一度だけスキップする

ラベル構文

```
label :
for(let i = 0; i < 5; i++){
    if(i > 2){
        break label;
    }
}
```

- ラベル名 :

  とすることでプログラムの任意の行に移行するラベルを付けられる

- ラベル付きbreak(continue)文⇐Cはない…

  break ラベル名として任意の行へループから抜けることが可能

- goto構文⇐Cの代替

  go to ラベル名をラベル付きbreak文の代わりとして使用できるが…

  ➡ 非推奨

for文のネスト ~多重ループ~

```
for(let i = 0; i < 5; i++){
    for(let j = 0; j < 3; j++){
        console.log(i * j);
    }
}
```

- 行列的な処理をするために多重ループというものが存在

  for文を入れ子構造にして中のforループから順に処理

- 右上の例では…

  0×0⇒0×1⇒0×2➡1×0⇒1×1⇒1×2➡2×0…

  のようにjからインクリメントされ、

  iの条件が満たされるまで繰り返しになる

演習４“制御構文”③

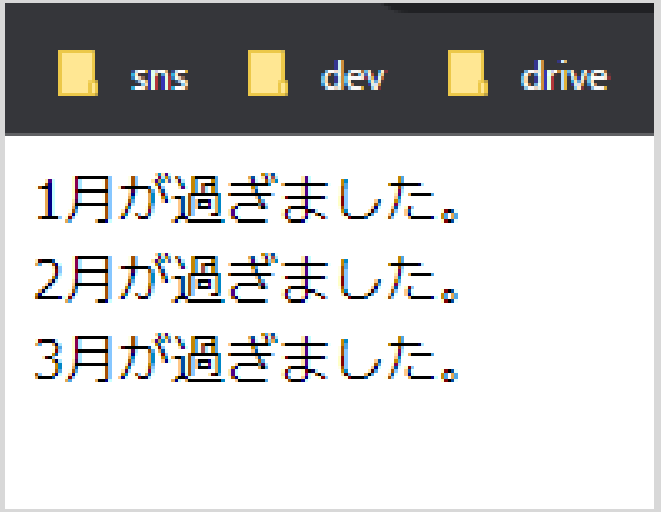
１．for文で1～12の月を1秒おきにループさせ、
　　一か月が一秒で過ぎる時空を歪めるプログラムを作成する

2. 多重ループを用いて＊で
　図形を作ってみる

```
function sleep(msec) {
    return new Promise(function(resolve) {
        setTimeout(function() {resolve()}, msec);
    });
}
async function brackhole(){
    //ここに処理を書く
    //await sleep(ms)で処理をストップできる
}
brackhole();
```

演習４“制御構文”③　回答

```
function sleep(msec) {
    return new Promise(function(resolve) {
        setTimeout(function() {resolve()}, msec);
    })
}
async function brackhole(){
    for(let month = 1; month < 13; month++){
        await sleep(1500);
        document.writeln(`${month}月が過ぎました。<br>`);
    }
}
brackhole();
```



演習４“制御構文”③　回答

```
for(let i = 0; i < 8; i++){
    let str = "";
    for(let j = 0; j < 8 - i; j++){
        str += "*";
    }
    document.writeln(str);
}
```

オブジェクト指向の様々な繰り返し構文

- for~in文
  オブジェクトの各要素に対して繰り返し処理を行う構文

- 仮変数に一時的にオブジェクトのキーを格納

- オブジェクトなどはオブジェクトの章で詳述

```
for(仮引数 in オブジェクト){
    //一つずつキーを取り出し反復
}
```

オブジェクト指向の様々な繰り返し構文

- for~of文

  配列の各要素に対して繰り返し処理を行う構文

- 仮変数に一時的に配列の要素が格納される

  →for~ofはキーに対し、for~inは要素

- 配列はオブジェクトの章で詳述

```
for(仮引数 in 配列){
    //一つずつ要素を取り出し反復
}
```

オブジェクト指向の様々な繰り返し構文

- foreach文
  C#などではfor~inをforeach~inで用いる

- オブジェクトやリスト（配列）に対して各要素ごとに反復処理

- オブジェクトの章で詳述

```
foreach(仮変数 in オブジェクト){
    //一つずつ要素を取り出して反復
}
```

オブジェクト指向の様々な繰り返し構文

- phpのfor~as構文や Visual BasicのFor Each ~ Next構文など
  オブジェクト指向型の言語には様々な繰り返し構文が

- 基本はfor文でカバーできるが、知っておくと便利

- 参考
  https://ja.wikipedia.org/wiki/Foreach%E6%96%87

try-catch-finally文

```
try{
    //処理
}catch(/*変数に例外を受け取る*/){
    //例外処理
}finally{
    //最終処理
}
```

- 開発時に想定外のエラーに遭遇することも

  ➡ 例外処理が有効

- tryで処理

  catchでtryで発生した例外の処理

  finallyで例外に関わらず最終的に実行される処理

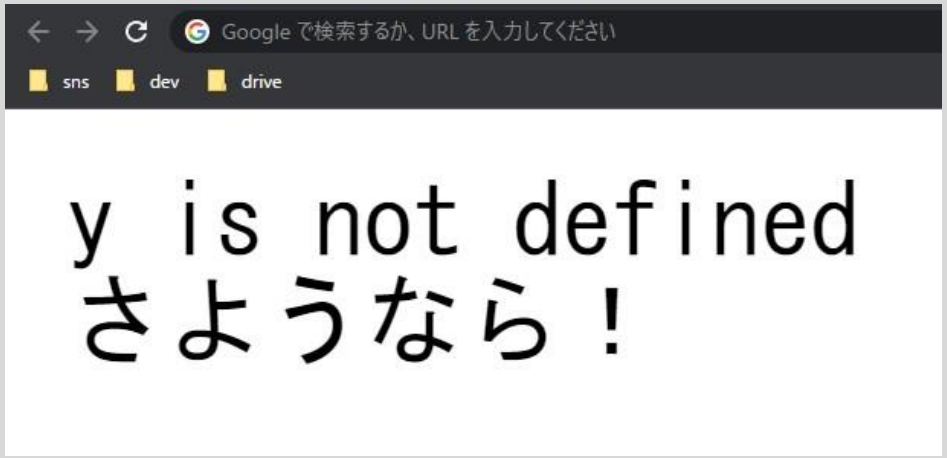
- 例外のデバッグにも使用できる

演習４“制御構文”④

定義していない変数をわざとtry文に登場させ、
　　故意的にcatch文の処理を行うプログラムを作成する

演習４“制御構文”④　回答

```
try{
    let x = 2;
    let result = x + y;
}catch(e){
    document.writeln(e.message);
}finally{
    document.writeln("さようなら！");
}
```



制御構文のまとめ

- 制御構文は構造化プログラミングの基本
- 順次、選択、反復の3つの種類がある
- 選択・・・if文やswitch文
- 反復・・・for文やwhile文

参考文献

- https://eng-entrance.com/linux-shellscript-variable
- https://xtech.nikkei.com/it/atcl/column/14/091700069/091700002/
- https://wa3.i-3-i.info/diff446programming.html
- http://www.b.s.osakafu-u.ac.jp/~hezoe/pro/chapter5.html
- https://kanda-it-school-kensyu.com/php-super-intro-contents/psi_ch09/psi_0905/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch06/jbi_0606/
- https://itmanabi.com/structured-objectoriented-prog/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0704/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0703/
- https://kanda-it-school-kensyu.com/java-basic-intro-contents/jbi_ch07/jbi_0702/