



多言語 プログラミング

～プログラミングにおける骨格～



概要

～まえがき～

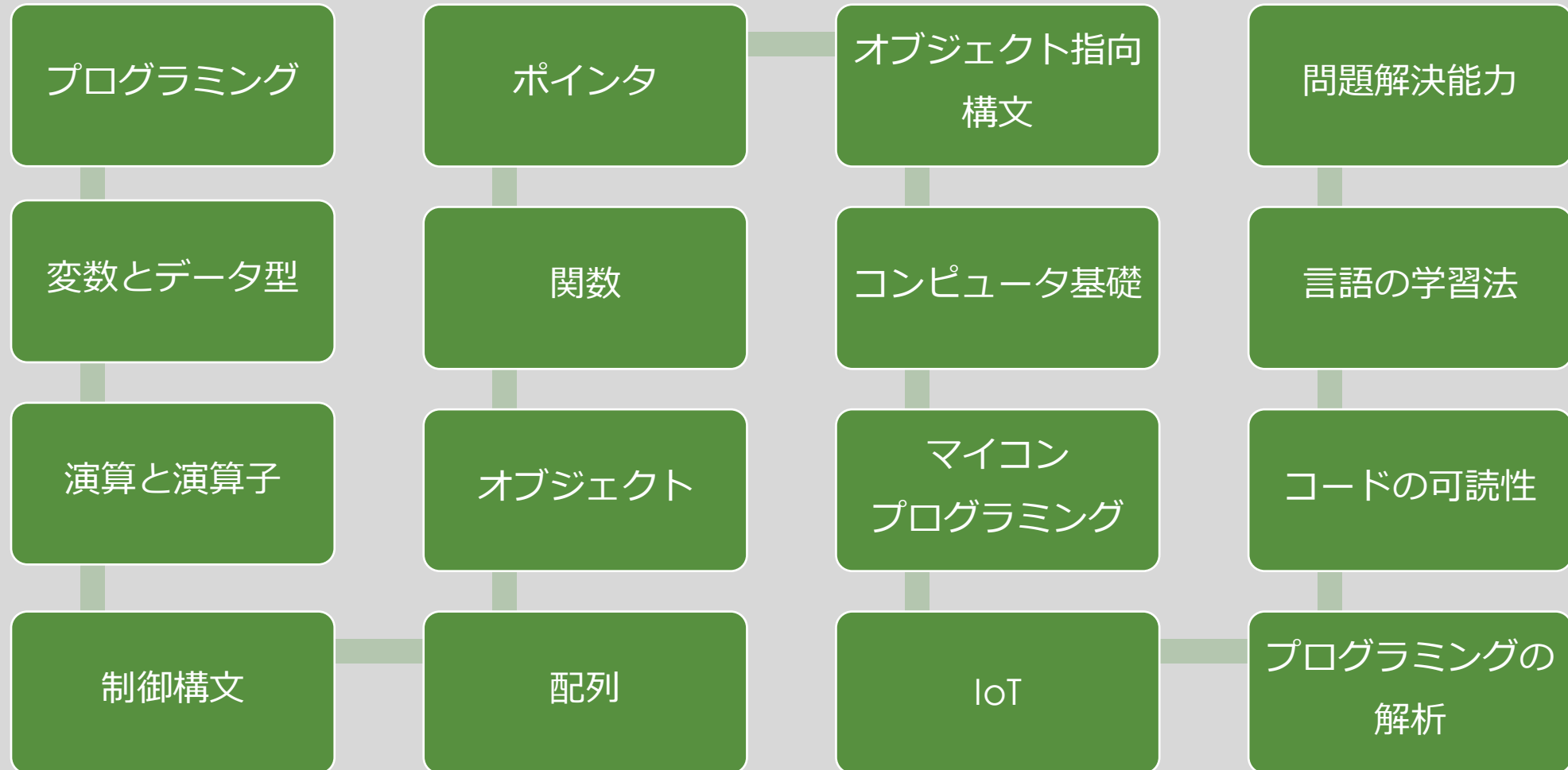
講義目的

- 多言語に渡るプログラミングを独学する術を身に着ける
- 基本的な用語や技術を学び、活かせる力を身に着ける
- マイコンやウェブなど、広い範囲に適応できる人材育成

進め方

- 重要な章ごとにまとめて講義
- 1 日（2時間） 1～2章分を進めたい
- 各章ごとに演習がある

カリキュラム



進め方

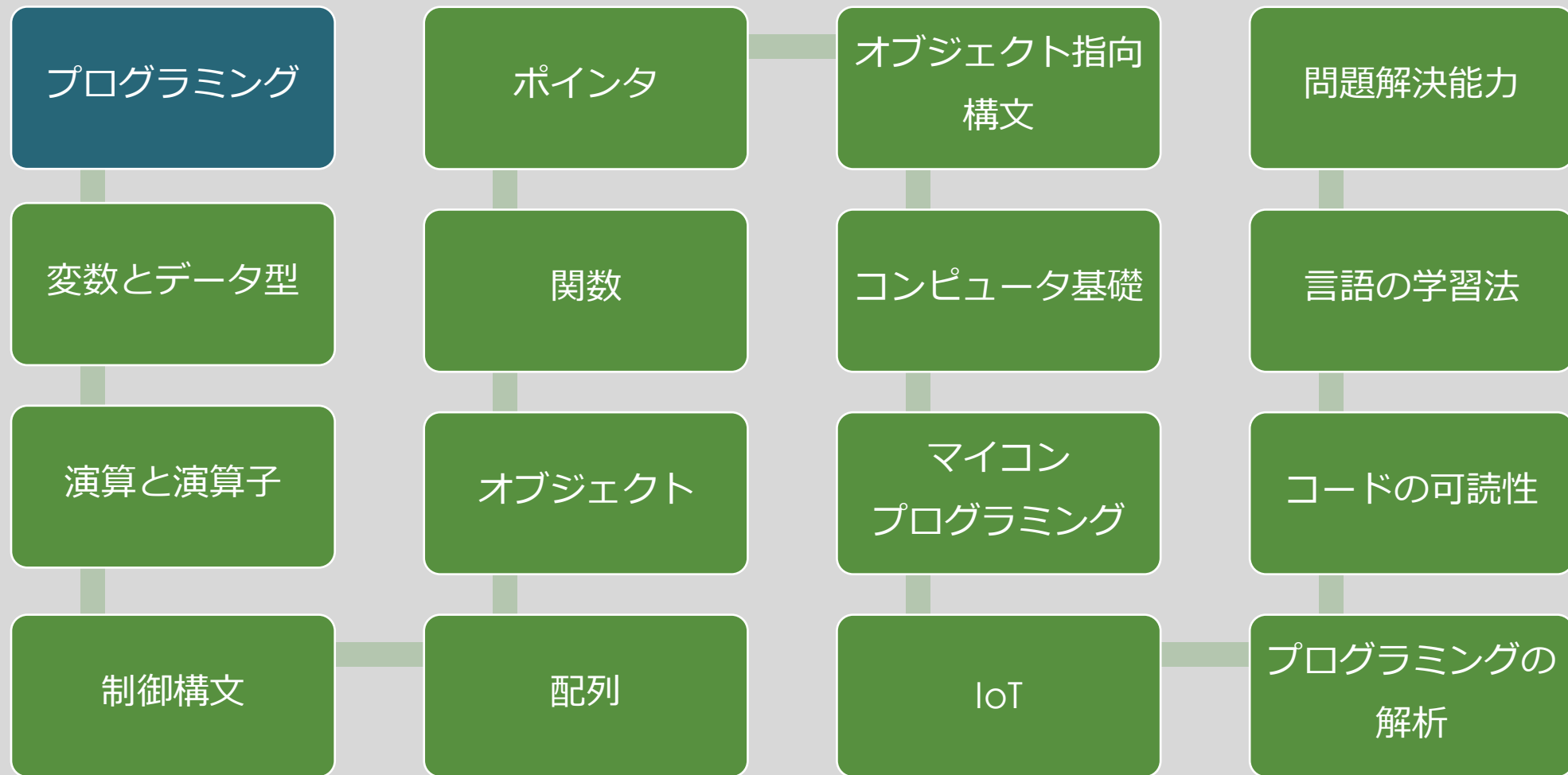
- プログラミングだけではなく、**設計**等の講義も行う
- 著者の**サイト**を用いた説明もある
- かなり分かりやすく**簡略化**しています。頑張ってください^^



プログラミングとは

～基本のキ～

進度



プログラミングどんなもの？

- プログラミングにはさまざまな種類が…
ウェブ、マイコン、ゲーム…etc
- 大まかには2種類
オブジェクト指向と構造化プログラミング（後述）
- 骨格は基本的に共通
関数や変数など、基本的な骨格がある

プログラミング言語

- **言語**は多数ある

C, C++, C#, JavaScript, VisualBasic, Python...etc

- 本教材では**JavaScript**とArduinoによる**C++**を使用

※その他にも様々な言語での文法や解析法を紹介

基本的な骨格とは？

- 冒頭でライブラリの宣言
- メインループで処理、UI等の場合ではイベントを処理
- サブルーチン（関数）にて繰り返し処理

…etc

```
#include <stdio.h>

int main(void){
    printf('Hello World');
    return 0;
}
```

```
void setup(){
    pinMode(2, OUTPUT);
}

void loop(){
    digitalWrite(2, HIGH);
    delay(200);
    digitalWrite(2, LOW);
    delay(200);
}
```

演習 1 “プログラミングとは”

- JavaScript動作確認

1. VSCodeの作業フォルダを作成

2. スクリプトで“Hello World”を表示させる。

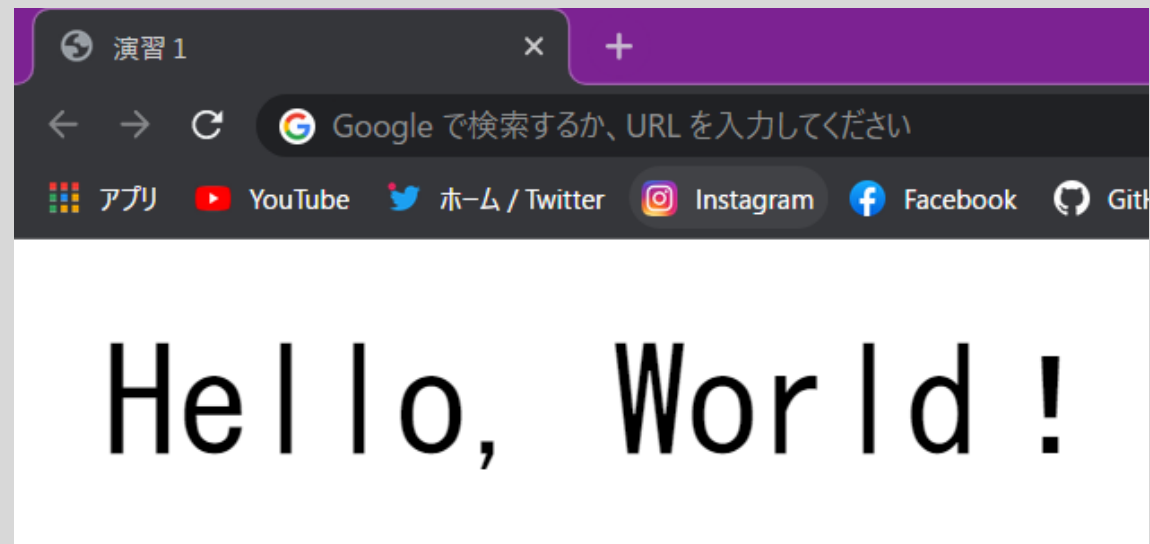
演習 1 “プログラミングとは”


- JavaScriptの書き方
 - htmlファイルの<script>タグ内に記述する
- ディスプレイさせる関数は、document.writeln()を使用
- 文末には;(セミコロン)を付ける

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;
charset=UTF-8" />
<title>Hello, World! </title>
</head>
<body>
<pre>
<script type="text/javascript">
//ここにスクリプトを記述
</script>
<noscript>JavaScriptが利用できません。</noscript>
</pre>
</body>
</html>
```

演習 1 “プログラミングとは” 回答

```
document.writeln('Hello, World !');
```

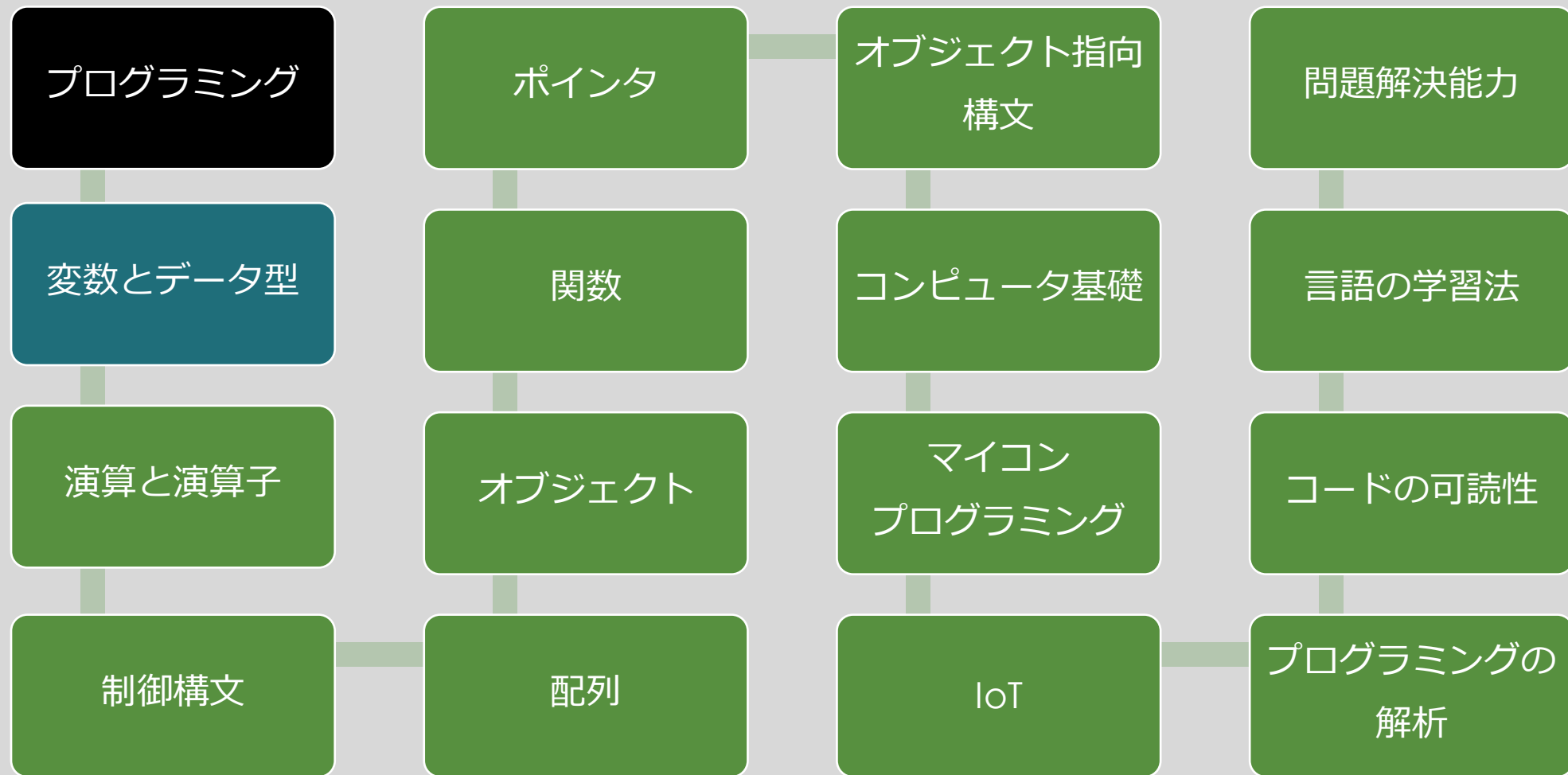




変数とデータ型

～プログラミングで使用する箱～

進度

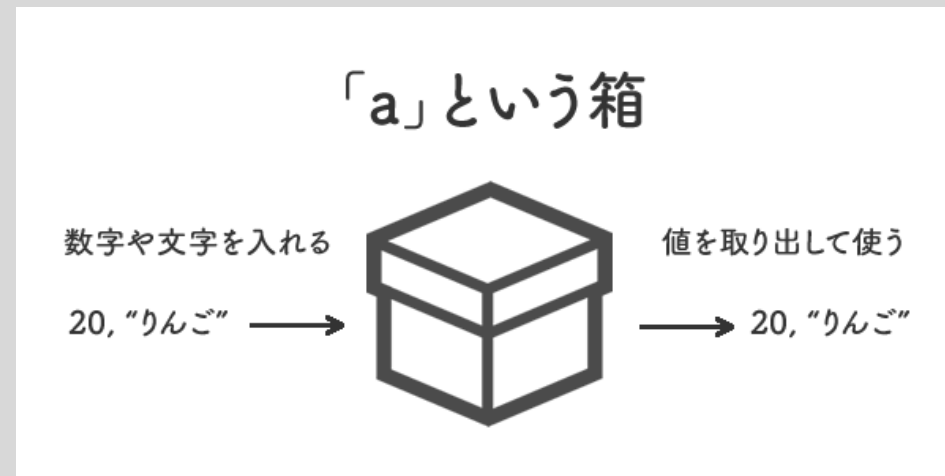


変数

- データを入れる箱

- 入れるデータによって種類がある

- 箱をひとまとめにしたもの → 配列



基本要素

- 宣言

変数を**作成**すること

- 初期化

変数の作成とともに**数値を割り当てる**こと

- 代入

変数の値を**上書き**すること

宣言

```
let x;
```

- どのような言語でも**基本同じ**
(python等では宣言が暗黙)

```
int x;
```

- **(修飾子) + 型 + 変数名**で宣言する
- 変数名は**予約語**以外なら原則何でも可

初期化

```
let x = 100;
```

```
int x = 100;
```

- 宣言とともに値を割り当てる
(Pythonでは宣言と初期化がセット)

```
int x = 'a';
```

- 型 + 変数名 = 初期値で宣言 + 初期化をセットで
- 初期値の型が一致している必要がある (詳細はp13にて)

初期化



- 初期化をしないと...

ランダムに適当な値“不定値”が割り当てられる

javascriptでは“undefined”と出る

予測していない結果になる可能性がある



- 基本的に宣言と初期化はセットでする

→初期化しないで使用するのは×

代入


- 変数を上書きして内部の値を更新する

```
let x = 100;
```

```
x = 200;
```

- 宣言後、任意の場所で 変数名 = 値
- = は実は演算子（詳細は演算子のところで）

データ型とは？

- 変数に入るデータは予め決めておかなければならない
  予め決める言語を静的型付け言語という
 - 整数（int）型、文字（char）型など多数ある
- ※ var型やlet型は直接的なデータ型ではない


データ型とは？

。様々なデータ型一覧

言語によってばらつきがあるが、基本的には以下の表

分類	名前	例
論理型	bool	True, false
文字型	char	a, b, c
文字列型	string	"abc"
整数型	int	1, 333
浮動小数点型	float, double	0.5, 0.0093
配列型	array	[1, 2, 3]
オブジェクト型	object	{x:1, y:2, z:3}

var型とlet型は？～動的型付け～

- **variety**（多様）からきている
JavaScriptでは**var**や**let**を用いて変数を使用
- **自動**でデータ型が判定される
 自動で割り当てられる言語を**動的型付け**言語という
- varは**宣言の重複**が可(上書きされる)、letは不可
- var型は後述の**スコープ**が特殊なので基本は**let**を使用

データ型の使い分け

- **サイズ**による使い分け

変数の**データ型**には**サイズ**がある

ex) int : 32bit、char : 16bit..etc 言語によってばらつきがある

- **オーバーフロー**

サイズを超えると**エラー**が起きたり、**予想しない結果**が…

※使う言語によるサイズの確認は**必須**

スコープとは？

変数には使える**範囲**がある

- **グローバル**変数

ファイル全体が有効範囲。
使いすぎるとメモリが..

- **ローカル**変数

その関数の内部のみ。

保持するにはstaticなどの修飾子を使用

※詳細は関数の章で説明

グローバルスコープ：スクリプト全体で有効

```
var global_Data = 'hoge' ;
```

グローバル変数

ローカルスコープ：関数の中でだけ有効

```
function foobar() {  
    var local_Data = 'hoge' ;  
}
```

ローカル変数

...

修飾子とは？

- オブジェクトやスコープに関わる変数の権限のようなもの
詳しくはオブジェクトの章で説明
- クラス間の変数にアクセスできないようにスコープ（アクセス）
を調節できたりする

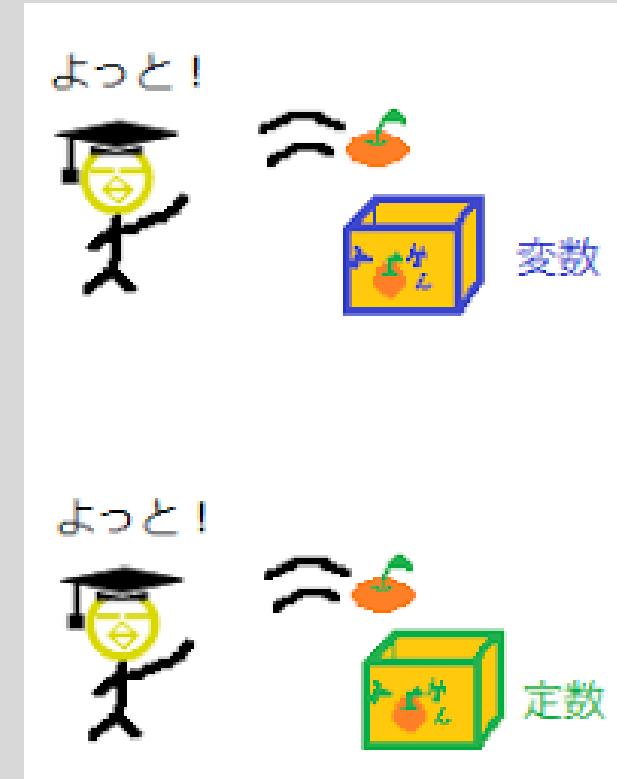
※ 言語によっては若干のばらつきがあるので注意が必要

修飾子の種類

- 変数のアクセス権限を宣言できる
`private` や `public` など...
- 変数の保持に使用
`static` など...
- ほかにも様々なものがある

定数

- 変数とは別に、数値に命名できる
- `const`修飾子を使用し、`const 定数名 = 値`
- 静的型付けでは、`const 型名 定数名 = 値`
とすることが多い
- 定数は変更（再代入）することができない



演習 2 “変数とデータ型”

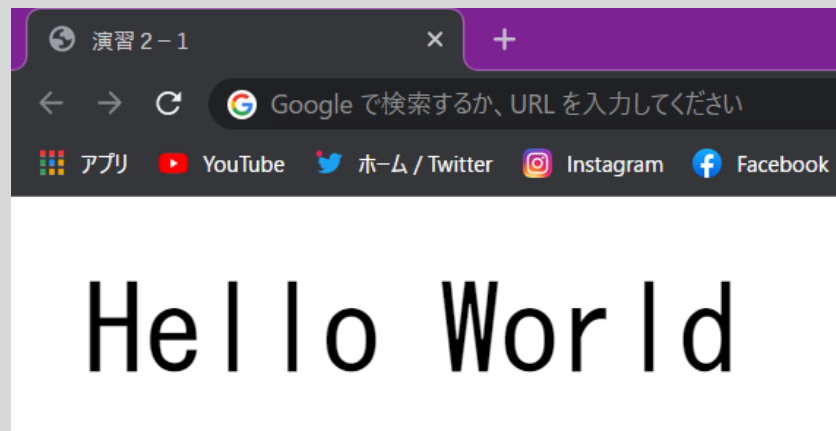
1. `let`型の変数を用いてHello Worldを表示させる
2. `var`型と`let`型の違いを確認する
3. 変数のメモリとスコープについて確認する
→関数の章で確認するため、回答を見て確認

演習 2 “変数とデータ型”① 回答

- 。宣言と初期化を使う方法と、宣言しその後代入という方法がある

```
let str = 'Hello World';  
document.writeln(str);
```

```
let str;  
str = 'Hello World';  
document.writeln(str);
```

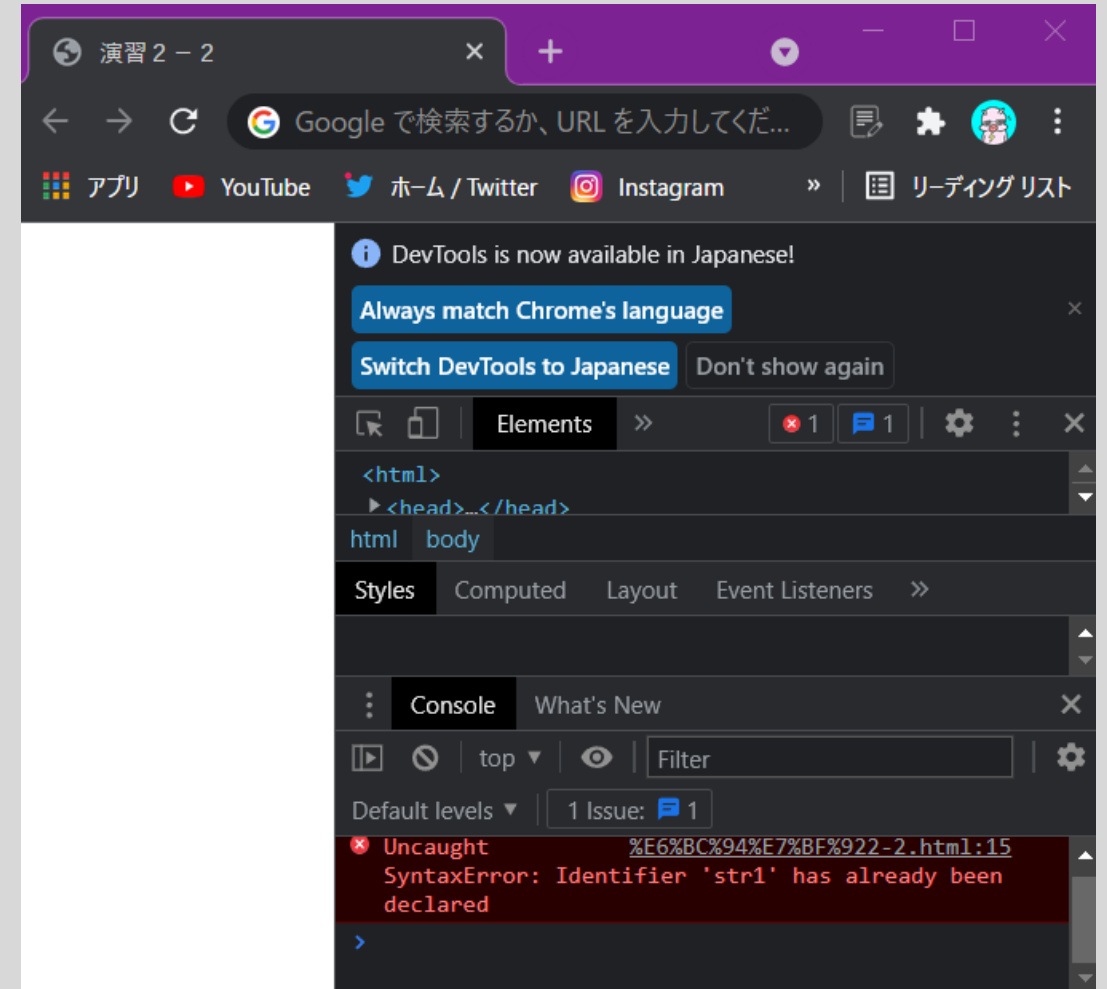


演習 2 “変数とデータ型”②

- **var**型と**let**型をそれぞれ再宣言するテストコードを実行し、
両方を実行して確認する

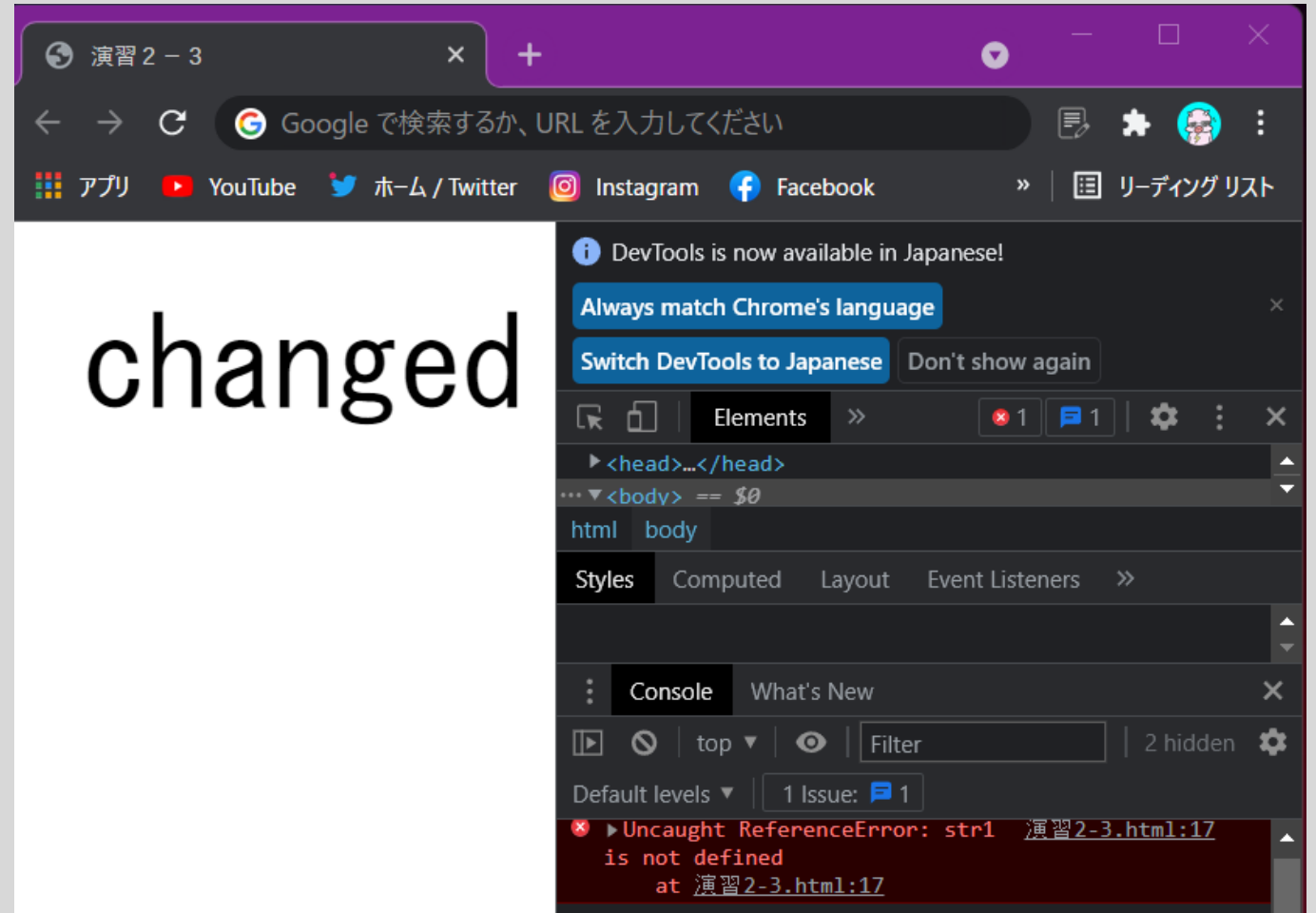
演習 2 “変数とデータ型”② 回答

```
var str = 'Hello World';  
var str = 'Hello JS';  
document.writeln(str);  
  
let str1 = 'Hello World';  
let str1 = 'Hello JS'  
document.writeln(str1);
```



演習 2 “変数とデータ型”③ 回答

```
let str = 'Hello World';
function test(){
  str = 'changed';
  let str1 = 'Hello Java';
}
test();
document.writeln(str);
document.writeln(str1);
```



ここまでのまとめ

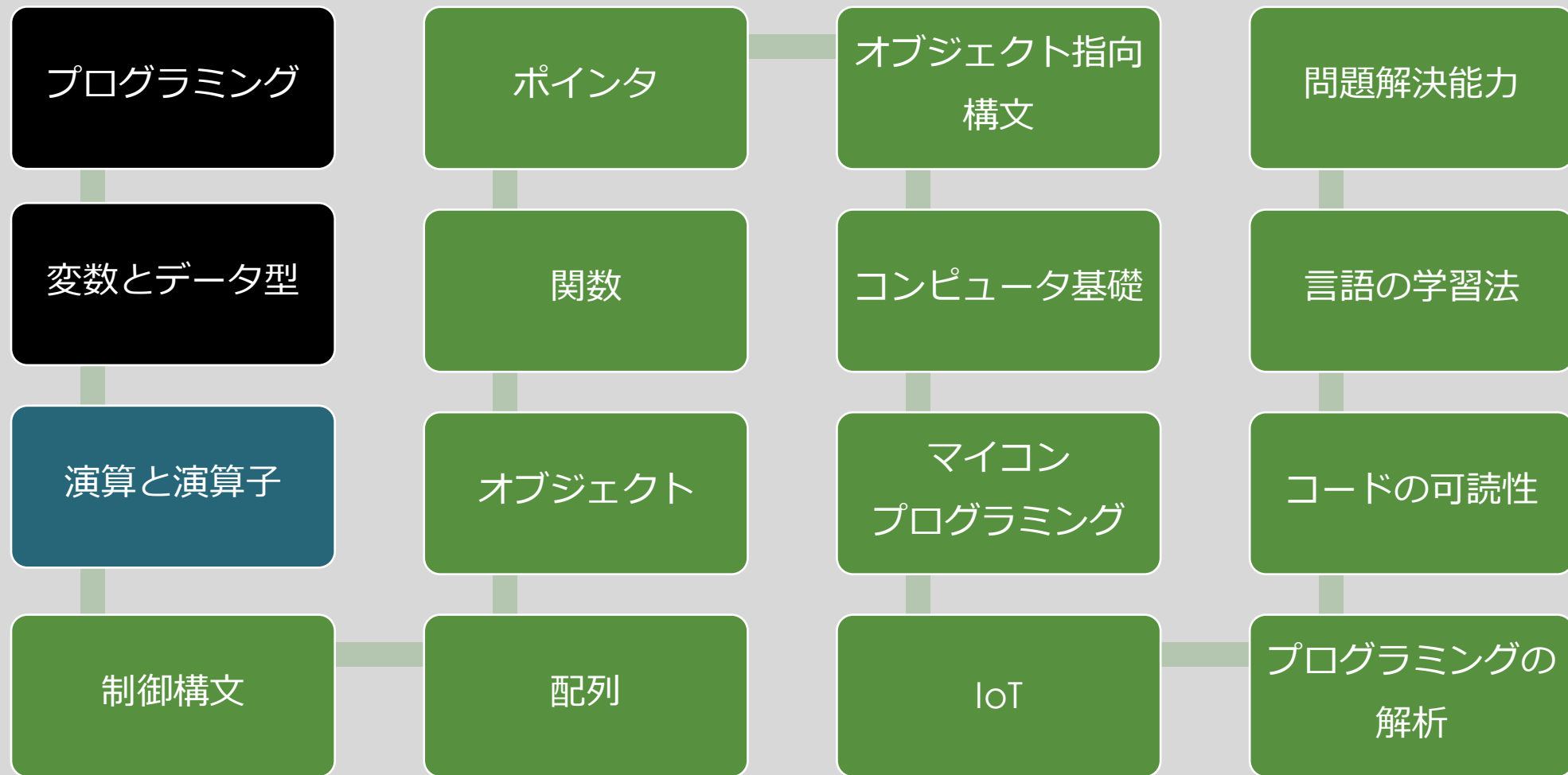
- プログラミングは**基本的な骨格**がある
- 変数はデータを入れる箱であり、**宣言**、**初期化**、**代入**などで操作する
- 変数は**スコープ**や**データ型**を確認する必要がある



演算と演算子

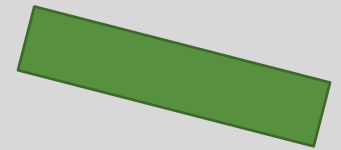
～プログラムの演算～

進度



演算の方法

- プログラミングは、**演算の繰り返し**で処理をする
- 演算には、**演算子**というものをを用いる
- 演算子の**優先順位**によって、多様な演算を実現する



演算子の種類

- 演算子にはさまざまな種類がある
- 算術演算子、代入演算子、比較演算子、
論理演算子、ビット演算子
- ↑が基本の演算子で、どの言語にもほとんど共通している

算術演算子

- 数学的な演算を行う演算子

```
let x = 5;  
let y = 3;  
x = x + 3;    //x : 8  
x = x * y;    //x : 24
```

- 加減算や乗除、剰余を求めるものがある

- 記法は数学と同じように、値 演算子 値 として計算を行う（2項）

1 + 2

基本的な算術演算子

- 数学的な演算を行う演算子

- べき乗演算子については
言語によりけり

演算子	概要	例
+	加算	$3 + 5$ (8)
-	減算	$5 - 2$ (3)
*	乗算	$5 * 2$ (10)
/	除算	$5 / 2$ (2.5)
%	剰余	$5 \% 2$ (1)
**	べき乗	$5 \wedge 2$ (25)

- 演算結果は基本的に変数のデータ型へ型変換（後述）される

※ 言語によって違うので注意

算術演算子（文字列の結合）

- 加算演算子を用いた文字列の結合が可能な言語が存在
- “文字列” + “文字列” として文字列を結合
※PHPの場合、“文字列” . “文字列” として結合が可能
- 言語によって型の制約があるため注意
※C#などの場合、文字列と整数型の加算は不可…etc

```
<?php
$str1 = 'Hello';
$str2 = 'World';

//HelloWorld
$str = $str1 . $str2;
?>
```

```
let str1 = 'Hello';
let str2 = 'World';
let str = str1 + str2; //HelloWorld
```

インクリメント/デクリメント演算子

- 1 を加算/減算するときに便利な演算子

```
let x = 2;  
x++;      //x : 3  
++x;      //x : 4
```

- インクリメントは'++'、デクリメントは'--'で記述
++は "+ 1"と同義、--は"- 1"と同義

- 記法としては、変数名 演算子 又は 演算子 変数名 とするだけ
※演算子の位置により演算の順序が異なる（次頁）

インクリメント/デクリメント演算子の順序

- 後置演算

演算対象の変数を処理した後、演算を行う

```
let x = 2;  
let y = x++;  
//yに代入 → xが3になる  
document.writeln(x); //3  
document.writeln(y); //2
```

- 前置演算

演算対象の変数を演算した後、処理を行う

```
let x = 2;  
let y = ++x;  
//xを加算 → yに代入  
document.writeln(x); //3  
document.writeln(y); //3
```

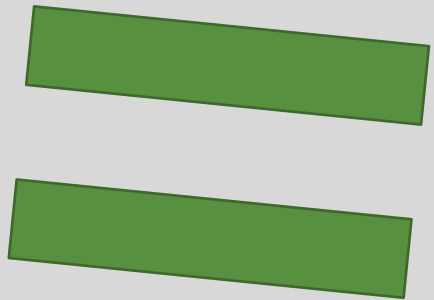
単項算術演算子

- 1 つの変数に対し演算を行う演算子
- 符号や論理の反転を行う

演算子	概要	例
+	変数(整数)の値を出力	+a (aがそのまま)
-	変数(整数)の符号を反転	-a (aの符号が反転)
!	変数(論理)の否定(not)	!a (aが0ならば1, 1ならば0)
~	変数(ビット)の1の補数	~a (aのすべてのビットを反転)

代入演算子（単純代入演算子）

- 演算した結果や値を、変数に設定（**代入**）するための**演算子**
→ "**=**" : 代入については前章参照
- 代入するには**型が一致**してる必要がある
- **参照による代入**と**値による代入**、**分割代入**がある
※ 参照による代入についてはオブジェクトの章で後述



代入演算子（複合代入演算子）

- 算術演算子やビット演算子と代入演算子が連動した演算子
- 言語によってはばらつきが...

複合代入演算子の種類

演算子	概要	例
<code>+=</code>	加算したものを代入	<code>x += 2 (x = x + 2)</code>
<code>-=</code>	減算したものを代入	<code>x -= 3 (x = x - 3)</code>
<code>*=</code>	乗算したものを代入	<code>x *= 2 (x = x * 2)</code>
<code>/=</code>	除算したものを代入	<code>x /= 5 (x = x / 5)</code>
<code>%=</code>	剰余を計算し代入	<code>x %= 3 (x = x % 3)</code>
<code>&=</code>	論理積を計算し代入	<code>x &= 4 (x = x & 4)</code>
<code> =</code>	論理和を計算し代入	<code>x = 5 (x = x 5)</code>
<code>^=</code>	排他的論理和を代入	<code>x ^= 3 (x = x ^ 3)</code>
<code><<=</code>	左にシフト演算し代入	<code>x <<= 2 (x = x << 2)</code>
<code>>>=</code>	右にシフト演算し代入	<code>x >>= 4 (x = x >> 4)</code>
<code>.=</code>	文字列を結合(PHP)	<code>x .= 'end' (x = x . 'end')</code>

比較演算子

- 条件分岐などで使用される、**論理の真偽**を評価する**演算子**
- 左辺と右辺の値を比較して、**True(1)**または**False(0)**を返す
- 言語によってばらつきがあるが、基本は同じ

比較演算子の種類

演算子	概要	例
==	左辺と右辺が等しいか	5 == 5(True, 1)
!=	左辺と右辺が等しくないか	5 != 5(False, 0)
<	左辺より右辺が大きいのか	5 < 5(False, 0)
<=	右辺が左辺以上か	5 <= 5(True, 1)
>	右辺より左辺が大きいのか	5 > 3(True, 1)
>=	左辺が右辺以上か	5 >= 3(True, 1)
===	左辺と右辺がデータ型まで等しいか	5 === 5(True, 1)
!==	左辺と右辺がデータ型まで等しくないか	5 !== 5(False, 0)
? :	三項演算子	(x == y) ? y + 1 : y

論理演算子

- 論理演算子を用いる条件式を組み合わせて全体を評価する演算子
- 条件分岐等の制御構文で用いられることが殆ど
※詳細は制御構文の章で
- 式全体の評価を、ショートカットする演算も存在（後述）

論理演算子の種類

- 。言語によってほとんど同じ、以下のような**演算子**がある

演算子	概要	例
&&	左右の式が両方True(1)か	5 == 5 && 20 == 20(True, 1)
	左右どちらかの式がTrue(1)か	5 == 5 20 == 10(True, 1)
!	式全体の評価を反転	!(5 < 5)(True, 1)

論理演算子のショートカット演算

- 式を評価するとき、**左式の真偽のみ**で評価が決定する演算
→ 右式の評価は行われない
- **&&**でのショートカット演算
→ 左式がfalse(0)の場合、右式がどうであれ**false**になる
- **||**でのショートカット演算
→ 左式がtrue(1)の場合、右式がどうであれ**true**になる

演習 3 “演算子”①

1. 算術演算子を用い、半径3[m]の球面の表面積を求める
→円周率はMath.PIを用いる
2. 複合代入演算子を用い、任意の定数を4乗する

※論理演算子に関しては制御構文で演習

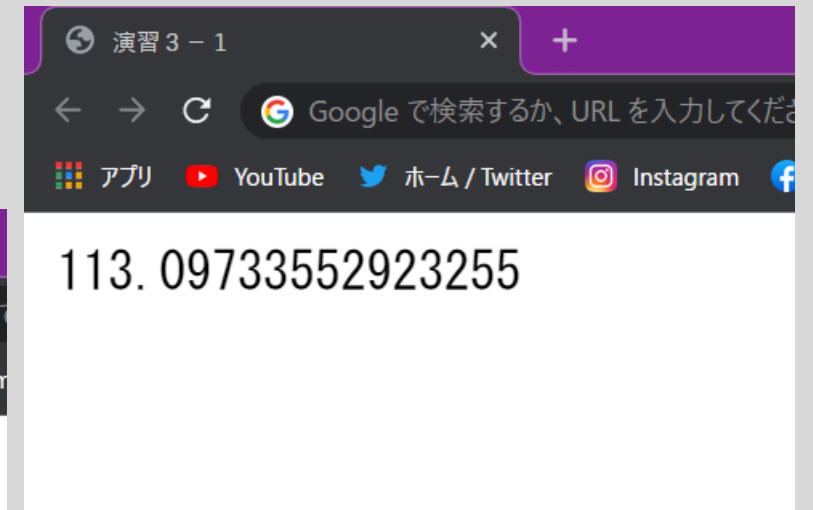
演習 3 “演算子”① 回答

1. 表面積 $4\pi r^2$ の公式を用いる

```
let r = 3;  
let area = 4 * Math.PI * r ** 2;  
  
document.writeln(area);
```

2. 演算子 $**=$ を用いる

```
let n = 5;  
n **= 4;  
  
document.writeln(n);
```



ビット演算子

- 。整数値を2進数で表し、2進数のビットに対して演算を行う
- 。マイコンにおいてシリアル通信やI2C, SPI通信で必要
- 。2進数で情報をやり取りすることは、非常に重要

ビット演算とは？ ～ビット論理演算子～

```
101
|001
-----
101
```

- 。論理和 (OR) → "|"

日本語では「A又はB」と表現される

演算の左右どちらかが1ならば、結果が1になる (A+Bと同じ)

```
101
&001
-----
001
```

- 。論理積 (AND) → "&"

日本語では「AかつB」と表現される

演算の左右どちらかが0ならば、結果が0になる (A×Bと同じ)

ビット演算とは？ ～ビットシフト演算子～

- 2進数のビットを右or左に、指定したビット数だけずらす演算
- ずらした後、端は0埋めに、はみ出しは切り捨てる
→ ビット数は変わらない
- 右シフト">>", 左シフト"<<"

101
----->>2
001

101
-----<<1
010

演習 3 “演算子”②

1. ビット論理演算子を用いて、ドモルガンの法則を確認する



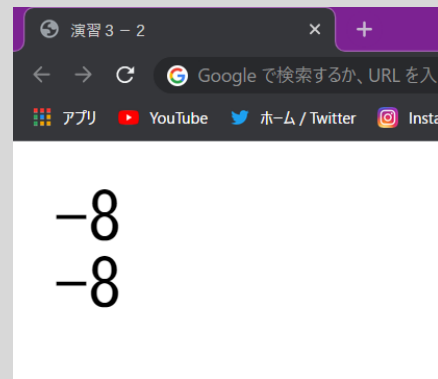
$$\sim A \ \& \ \sim B = \sim(A \mid B)$$

$$\sim A \mid \sim B = \sim(A \ \& \ B)$$

2. ビットシフト演算子を用いて、シフト演算が2の乗算になることを
確認する

演習 3 “演算子”② 回答

```
let x = 5;  
let y = 7;  
  
let result1 = (~x) & (~y);  
let result2 = ~(x | y);  
  
document.writeln(result1);  
document.writeln(result2);
```



```
let x = 5;  
let y = 7;  
  
let result = 5 << 3;  
  
document.writeln(result);
```



型変換とキャスト（Cなど）

```
int x = 2;  
int y = 3;  
// int/doubleの計算  
// 暗黙的に型が変換される  
double ave = (x + y) / 2.0;
```

暗黙的な型変換

→ 違う型同士で演算したとき、どちらかに型変換されて演算

- 暗黙的なため、随時どの型に変換されているか **把握しにくい**
→ **可読性**も下がってしまう…

型変換とキャスト（Cなど）

```
int x = 2;  
int y = 3;  
// (double) + (式)  
// 明示的にdouble型として演算される  
double ave = (double)(x + y) / 2;
```

明示的な型変換（キャスト演算）

→(型) + 式 or 値 とすることで明示的に型変換が可能

◦キャスト演算子

→キャスト演算において (型) をキャスト演算子と呼ぶ

演算子の優先順位

- 演算には優先順位が存在
- 複雑な演算で意識が必要
- 右図はJavaScriptの例

優先順位	演算子
高 ↑	かっこ (())、配列 ([])
	インクリ/デクリメント、単項算術演算子
	乗算 (*)、除算 (/)、剰余 (%)
	加算 (+)、減算 (-)、文字列結合 (+)
	シフト演算子
	比較演算子 (<、<=、>、>=)
	比較演算子 (==、!=、===、!==)
	AND(&)
	OR()
	論理積 (&&)
低	論理和 ()
	三項演算子 (?:)
	代入演算子
	カンマ (,) : クラスで使用

演算子の結合則

- 演算子を左か右で
結合するかを決定する
- 言語によって
基本的に同じ

結合性	演算子の種類
左→右	算術演算子
	比較演算子
	論理演算子
	ビット演算子
	かっこや配列
右→左	インクリ/デクリメント
	代入演算子
	単項演算子
	三項演算子
	deleteやtypeof等

変換指定（Cなど）

- printfやscanf関数などで、**文字列**に**数値**を埋め込むときに使用
- %ab.cd（abcは定数）のように使用する
 - a: **0フラグ**、数値の前の余白を**0**で埋める（オプション）
 - b: **最小フィールド値**、最低限の**表示文字数**（オプション）
 - c: **精度**、表示する**最小の桁数**（オプション）
 - d: **変換指定子**、データ型に対応した文字が必要

テンプレート文字列 (JavaScriptなど)

- 文字列へ変数を埋め込むときに使用する
- 文字列結合演算子(+)を使わずに、簡潔になる
- `${変数名}`として文字列に埋め込む
→ `` (バッククォート) で囲む

```
let name = 'Jin';  
let str = `こんにちは、  
${name}さん。`;
```

演習 3 “演算子”③

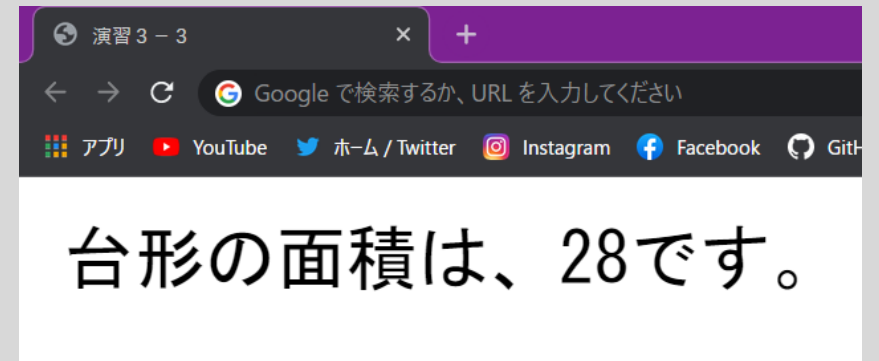
1. 上底3、下底4、高さ8の台形の面積を求め、
優先順位と結合性確認する
2. テンプレート文字列を用いて、1 の結果を文字列に埋め込む

演習 3 “演算子”③ 回答

```
let jotei = 3;  
let katei = 4;  
let takasa = 8;  
  
let result =  
  (jotei + katei) * takasa / 2;  
document.writeln(result);
```



```
let jotei = 3;  
let katei = 4;  
let takasa = 8;  
  
let result = (jotei + katei) *  
takasa / 2;  
document.writeln(`台形の面積は、  
${result}です。`);
```



演算子のまとめ

- 演算子には算術やビット、論理などさまざまな種類がある
- 代入演算子には単純と複合がある
- 演算子には優先順位や結合性があり、複雑な計算をするときに必要
- 変数は文字列に埋め込める