

# クラス①

2019/12/04

木南 貴志

# オブジェクト指向

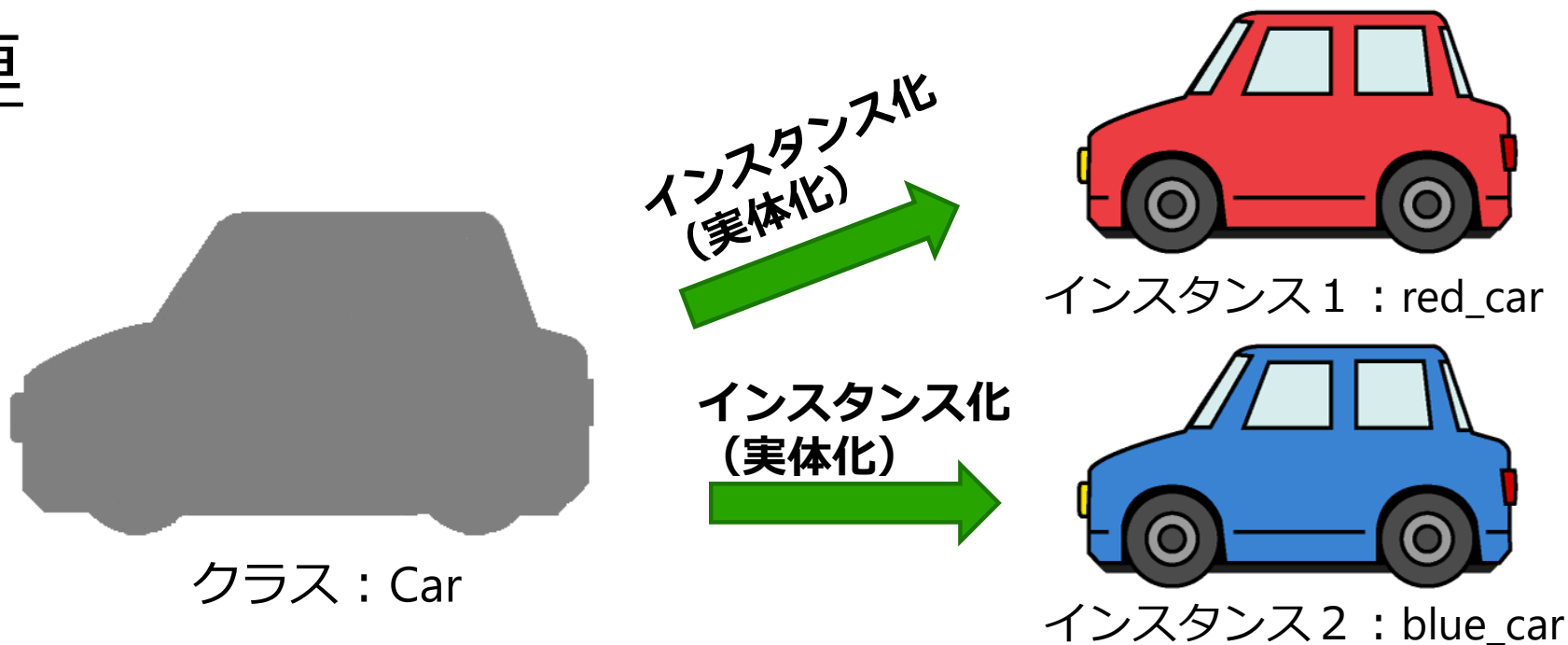
Pythonはオブジェクト指向の言語

**オブジェクト指向**：クラス、関数をオブジェクト（物）として扱う

**クラス**：オブジェクトを生成するための型

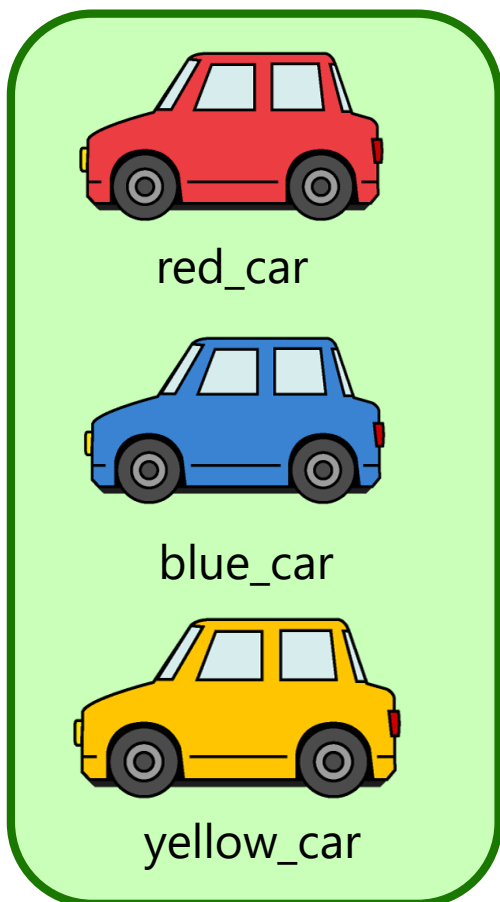
**インスタンス**：クラス（型）からインスタンスを生成（実体化）

例）車



# オブジェクト指向をする理由①

## •作業量を省略可能

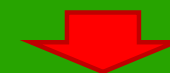


- **red\_car**  
red\_carに走る機能を付与  
red\_carに止まる機能を付与  
red\_carを赤色に設定
- **blue\_car**  
blue\_carに走る機能を付与  
blue\_carに止まる機能を付与  
blue\_carを青色に設定
- **yellow\_car**  
yellow\_carに走る機能を付与  
yellow\_carに止まる機能を付与  
yellow\_carを黄色に設定

red\_carで  
「走る」「止まる」  
機能を設定

blue\_carで  
「走る」「止まる」  
機能を設定

yellow\_carで  
「走る」「止まる」  
機能を設定

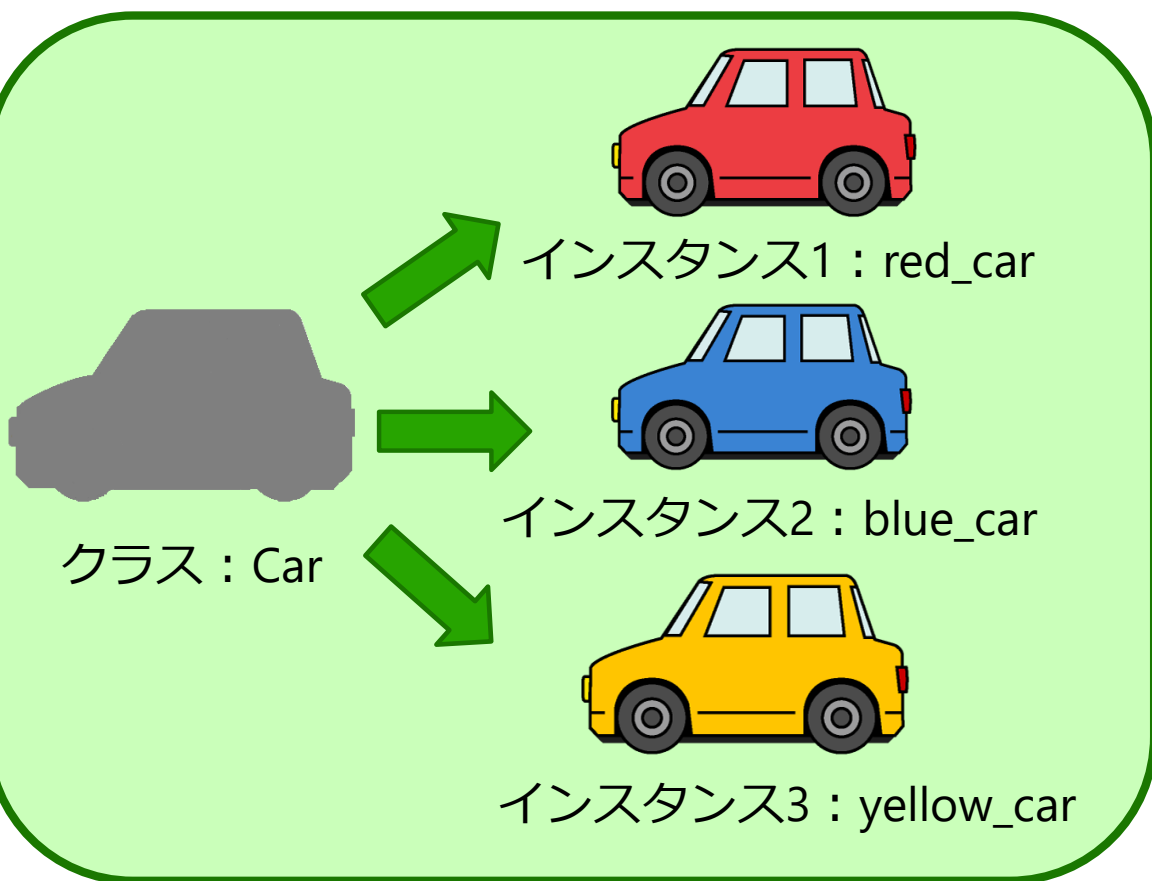


同じ設定を  
車ごとに行う必要あり

クラスを使用しない場合

# オブジェクト指向をする理由①

## •作業量を省略可能



クラスを使用する場合

- **クラスCar**  
走る機能を付与  
止まる機能を付与
- **Car ➡ red\_car**  
red\_carを赤色に設定
- **Car ➡ blue\_car**  
blue\_carを青色に設定
- **Car ➡ yellow\_car**  
yellow\_carを黄色に設定

どのインスタンス (red\_car, blue\_car, yellow\_car) にも共通する機能 (走る・止まる) はクラス (Car) で設定

➡ 作業量の省略が可能

# クラスを使用する時としない時

## 前ページの例をコードで比較

### ・クラスを使用しない場合

似たような  
コードを  
3回記述

```
red_car_run = "run"  
red_car_stop = "stop"  
red_car_color = "red"
```

```
blue_car_run = "run"  
blue_car_stop = "stop"  
blue_car_color = "blue"
```

```
yellow_car_run = "run"  
yellow_car_stop = "stop"  
yellow_car_color = "yellow"
```

red\_carを走らせたい時

```
print(red_car_run) #結果 run
```

車が共通して  
持つ機能は  
クラス内で記述

### ・クラスを使用する場合

```
class Car:  
    def __init__(self, color):  
        self.color = color  
    def car_run(self):  
        print("run")  
    def car_stop(self):  
        print("stop")
```

```
red_car = Car("red")  
blue_car = Car("blue")  
yellow_car = Car("yellow")
```

red\_carを走らせたい時

```
red_car.car_run() #結果 run
```

# クラスのコードの例①

## クラスを使用した時のコードの説明

```
class Car:
```

```
    def __init__(self, color):
```

```
        self.color = color
```

```
    def car_run(self):
```

```
        print("run")
```

```
    def car_stop(self):
```

```
        print("stop")
```

```
red_car = Car("red")
```

```
blue_car = Car("blue")
```

```
yellow_car = Car("yellow")
```

**class (classの名前):** でクラスを作成  
クラスの頭文字は大文字にするのがルール

# クラスのコードの例②

## コンストラクタ（初期化）

```
class Car:  
    def __init__(self, color):  
        self.color = color  
    def car_run(self):  
        print("run")  
    def car_stop(self):  
        print("stop")  
red_car = Car("red")  
blue_car = Car("blue")  
yellow_car = Car("yellow")
```

- ・ **def \_\_init\_\_**（コンストラクタ（初期化））  
def \_\_init\_\_ に書かれたコードはインスタンス作成時に必ず実行

※**\_\_init\_\_**のアンダーバーは二つであることに注意

- ・ **selfはインスタンス自身を示す**  
クラス内に作成する関数（メソッドという）の一つ目の引数は必ずself  
※selfについては次ページで詳しく解説

# self

## selfはインスタンスそのもの

selfを指定した場合（インスタンス変数）

```
class Car:  
    def __init__(self, color):  
        self.color = color
```

```
red_car = Car("red")  
print(red_car.color)
```

**結果**

red

上記ではself = red\_carなので  
red\_car.colorと指定すると  
red\_carの色の情報を引き出せる

selfを指定ない場合（ローカル変数）

```
class Car:  
    def __init__(self, color):  
        color = color
```

```
red_car = Car("red")  
print(red_car.color)
```

**結果（エラーが出る）**

'Car' object has no attribute 'color'

\_\_init\_\_でcolorを宣言しているが、selfを  
つけていない  
➡ red\_carに色情報は与えられていない



# self

## selfはインスタンスそのもの

selfを指定した場合（インスタンス変数）

一致

```
class Car:
    def __init__(red_car, "red"):
        red_car.color = "red"

red_car = Car("red")
print(red_car.color)
```

red\_carを生成する場合  
self に red\_car  
color に "red" が代入

結果

red

上記ではself = red\_carなので  
red\_car.colorと指定すると  
red\_carの色の情報を引き出せる

selfを指定ない場合（ローカル変数）

不一致

```
class Car:
    def __init__(red_car, "red"):
        color = "red"

red_car = Car("red")
print(red_car.color)
```

selfがないと呼び出せない

結果（エラーが出る）

'Car' object has no attribute 'color'

\_\_init\_\_でcolorを宣言しているが、selfを  
つけていない  
➡ red\_carに色情報は与えられていない

# クラスのコードの例③

## クラスを使用した時のコードの説明

```
class Car:  
    def __init__(self, color):  
        self.color = color  
    def car_run(self):  
        print("run")  
    def car_stop(self):  
        print("stop")  
red_car = Car("red")  
blue_car = Car("blue")  
yellow_car = Car("yellow")
```

### ・ **def メソッド():**

クラス内に作成された関数を**メソッド**

生成されたインスタンスから呼び出せる

※詳細は次ページより

# メソッド

red\_carのcar\_runメソッドを例に説明

```
class Car:
    def __init__(self, color):
        self.color = color
    ① def car_run(self):
        print("run")
    def car_stop(self):
        print("stop")
red_car = Car("red")
red_car.car_run()
```

- ① クラス内にcar\_runメソッドを作成
- ② インスタンスred\_carを生成  
\_\_init\_\_メソッド（初期化）を実行  
（selfにred\_car、colorに"red"を代入）
- ③ car\_runメソッドを呼び出し実行

# メソッド

red\_carのcar\_runメソッドを例に説明

```
class Car:
    def __init__(red_car, "red"):
        red_car.color = "red"
    def car_run(red_car):
        print("run")
    def car_stop(red_car):
        print("stop")
```

② red\_car = Car("red")  
red\_car.car\_run()

- ① クラス内にメソッドを作成
- ② インスタンスred\_carを生成  
\_\_init\_\_メソッド（初期化）を実行  
（selfにred\_car、colorに"red"を代入）
- ③ car\_runメソッドを呼び出し実行

# メソッド

red\_carのcar\_runメソッドを例に説明

```
class Car:
    def __init__(red_car, "red"):
        red_car.color = "red"
    def car_run(red_car):
        print("run")
    def car_stop(red_car):
        print("stop")
red_car = Car("red")
③ red_car.car_run()
```

- ① クラス内にメソッドを作成
- ② インスタンスred\_carを生成  
\_\_init\_\_メソッド（初期化）を実行  
（selfにred\_car、colorに"red"を代入）
- ③ **car\_runメソッドを呼び出し実行**

car\_runメソッドを呼び出す

結果

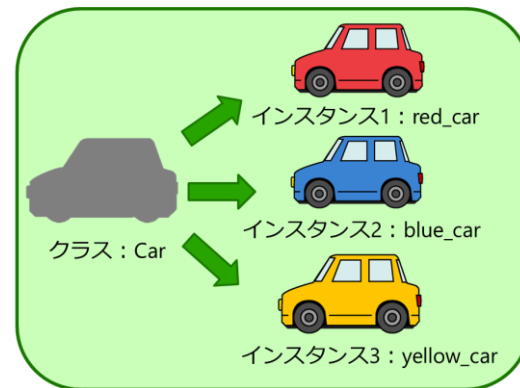
run

# クラスのコードの例④

## インスタンスの生成

```
class Car:
    def __init__(self, color):
        self.color = color
    def car_run(self):
        print("run")
    def car_stop(self):
        print("stop")
```

```
red_car = Car("red")
blue_car = Car("blue")
yellow_car = Car("yellow")
```



インスタンス名 = クラス名 ( \_\_init\_\_ の引数 )

• **red\_car = Car("red") の場合**

インスタンス作成時は必ず

`__init__` (コンストラクタ) を実行

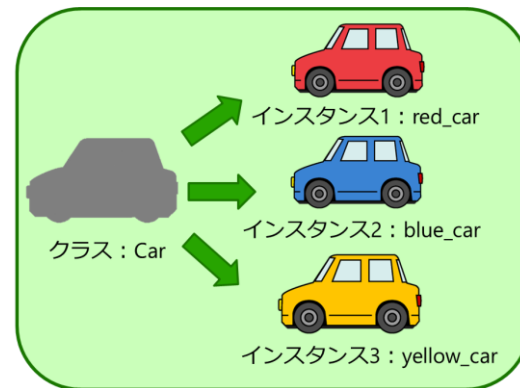
```
def __init__(self, color):
    self.color = color
```

# クラスのコードの例④

## インスタンスの生成

```
class Car:
    def __init__(self, color):
        self.color = color
    def car_run(self):
        print("run")
    def car_stop(self):
        print("stop")
```

```
red_car = Car("red")
blue_car = Car("blue")
yellow_car = Car("yellow")
```



インスタンス名 = クラス名 ( \_\_init\_\_ の引数 )

・ **red\_car = Car("red") の場合**

インスタンス作成時は必ず

`__init__` (コンストラクタ) を実行

```
def __init__(red_car, "red")
    red_car.color = "red"
```

**selfにred\_car  
colorに"red"が代入**

# 練習問題

1. class Carにクラクションを鳴らすメソッドhornを追加せよ

クラクションの音は"boooo"とする

ヒント：car\_run、car\_stopとほとんど同じ

2. インスタンスblack\_carを新たに生成する。black\_carで作成したメソッドhornを実行せよ

ヒント：出力結果は下記になる

```
boooo
```



# オブジェクト指向をする理由②

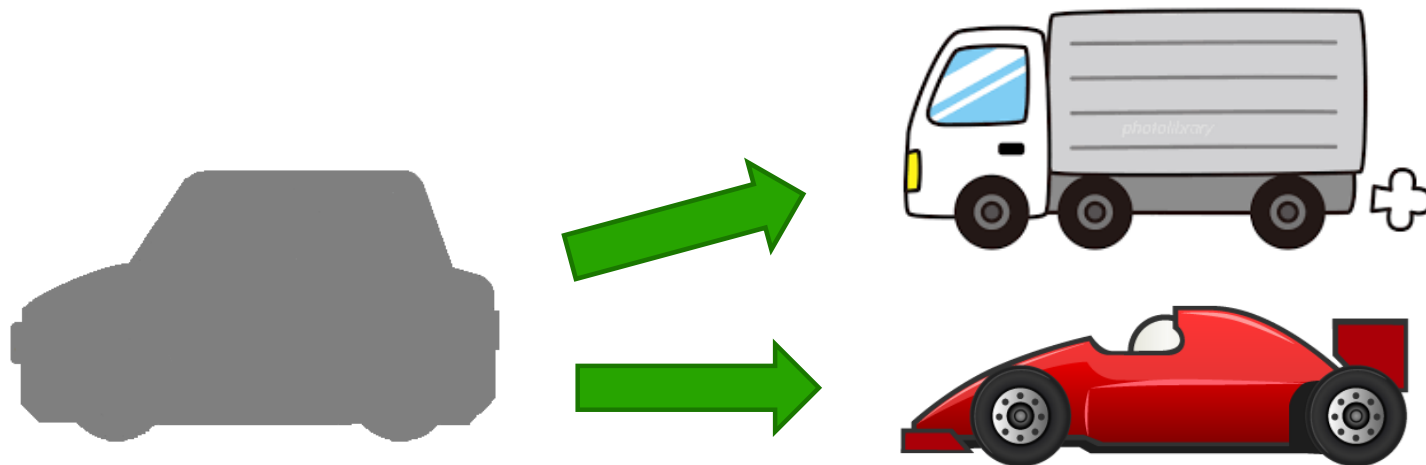
## 大人数で開発するときに便利（車を例に説明）

- ・「車」をあらかじめ用意
  - ➡ 「走る」「止まる」のコードを知らなくても車を使用できる
- ・「車」を用意しない
  - ➡ 「走る」「止まる」のコードを理解する必要あり
    - ➡ 正しく理解しないと「走る」「止まる」の部分を破壊する恐れあり

「車」をあらかじめ用意すれば基本的な機能の中身を考えず、追加したい機能のみに集中可能

# オブジェクト指向をする理由③

同じようなモノを作りやすい（車を例に説明）



レーシングカー、トラックのどちらも「走る」「止まる」は共通  
➡ 使用できる機能は再利用

**上手く再利用すれば作業効率が向上**

# オブジェクト指向をする理由③

同じようなモノを作りやすい（具体例）

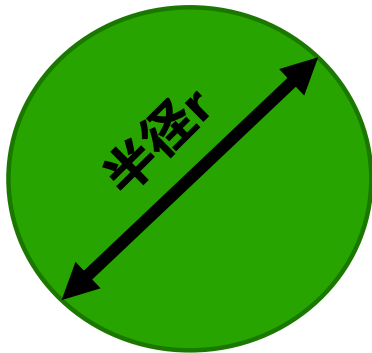


# クラス変数、インスタンス変数

クラス変数：クラス内にある変数（すべてのインスタンスに共通する変数）

インスタンス変数：メソッド内にある変数（個々のインスタンスで異なる変数）

例) 円

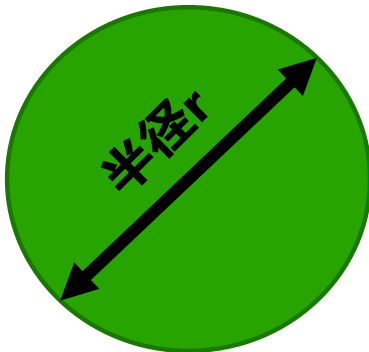


- ・ クラス変数（全ての円が共通した値をもつ変数）：  
円周率（ $=3.141592653589793$ ）
- ・ インスタンス変数（円によって異なる値を持つ変数）：  
半径

# クラス変数、インスタンス変数

前ページをコードで記述

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r
```



pi = 3.141592653589793  
をクラス変数（どんな円を作成する時も同じ値）として宣言

self.radius = r  
をインスタンス変数（作成する円によって変わる値）として宣言

## 練習問題②

---

1. class Circlesに円の円周を求めるメソッドを追加せよ  
※メソッド名はなんでもよい  
ヒント：計算した値を返すにはreturnを使用
2. 【半径2の円circle1】を生成して、上記で作成したメソッドを使用し円周を求めて画面上に出力させろ  
また、【半径3の円circle2】でも同様に実装せよ

# クラス変数、インスタンス変数、ローカル変数の比較

## クラス変数、インスタンス変数、ローカル変数の比較

### クラス変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle1 = Circles(2)  
print(circle1.pi)
```

#### 結果

3.141592653589793

### インスタンス変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle1 = Circles(2)  
print(circle1.radius)
```

#### 結果

2

### ローカル変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle1 = Circles(2)  
print(circle1.hoge)
```

#### 結果（参照できない）

'Circles' object has no attribute 'hoge'

# クラス変数、インスタンス変数、ローカル変数の比較

## 引数を2 ➡ 3 に変更したインスタンスcircle2を生成

### クラス変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle2 = Circles(3)  
print(circle2.pi)
```

### 結果

3.141592653589793

どのインスタンスでも同じ値

### インスタンス変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle2 = Circles(3)  
print(circle2.radius)
```

### 結果

3

インスタンスによって違う値

### ローカル変数の参照

```
class Circles:  
    pi = 3.141592653589793  
  
    def __init__(self, r):  
        self.radius = r  
        hoge = 1  
  
circle2 = Circles(3)  
print(circle2.hoge)
```

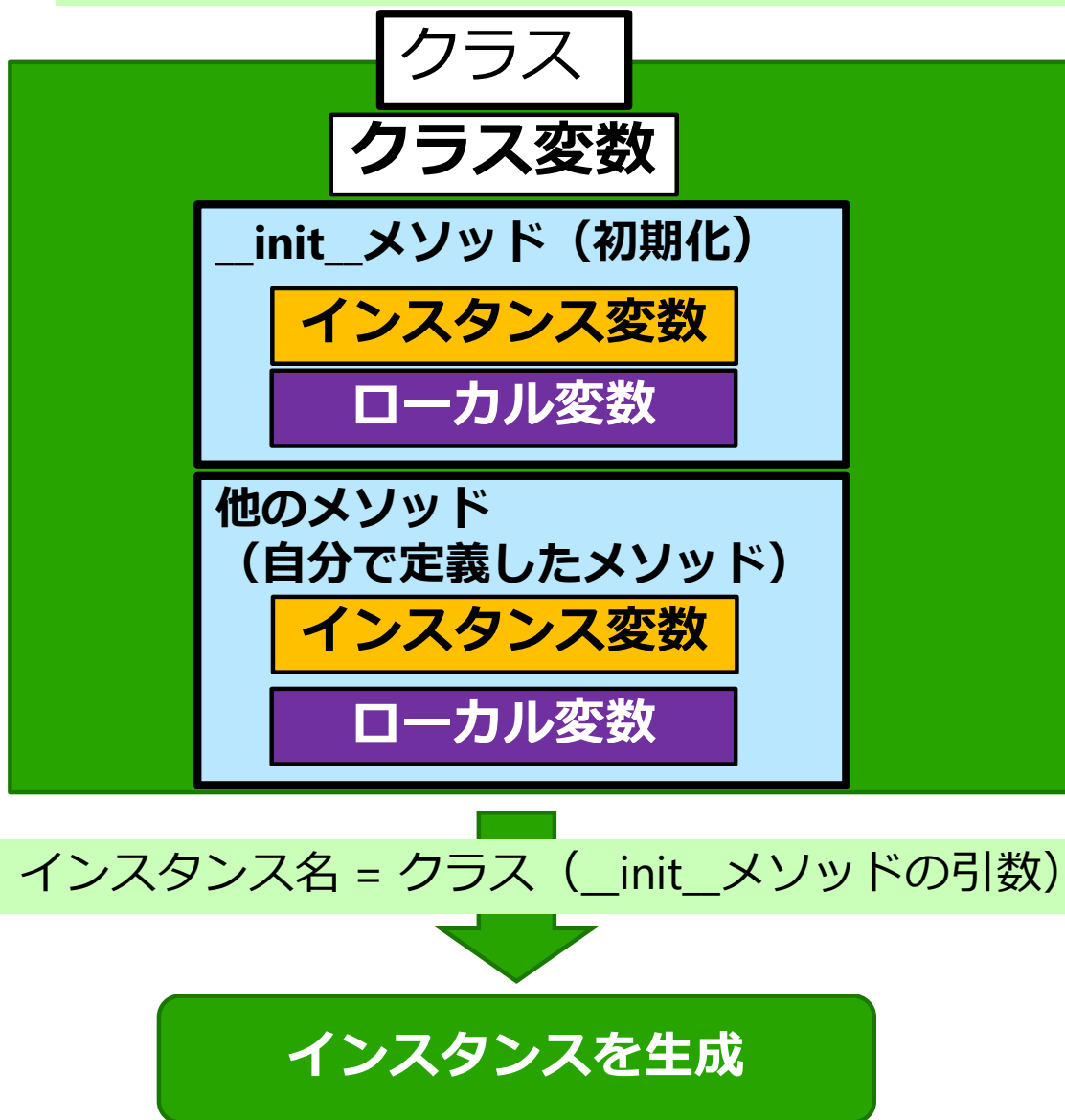
### 結果（参照できない）

'Circles' object has no attribute 'hoge'

メソッド（関数）内でのみ有効



# クラス まとめ



**クラス**：オブジェクトを生成するための型

**クラス変数**：どのインスタンスを生成する時も共通する値をもつ

**インスタンス変数**：生成するインスタンスによって異なる値をもつ

**\_init\_メソッド (初期化)**：インスタンス生成時に必ず実行されるメソッド

**インスタンス**：クラス（型）を実体化したもの

# クラス まとめ 例

Circles

`pi = 3.141592653589793`

`__init__`メソッド (初期化)

`self.radius = r`

`area`メソッド

`def area(self):`

`return self.pi * self.radius ** 2`

**クラス** : Circles

**クラス変数** : 円周率pi (どの円を作成するときにも共通する値)

**インスタンス変数** : 半径self.radius (作成する円によって値が変わる)

**\_\_init\_\_メソッド (初期化)** : インスタンス生成時、selfにcircle1、rに2を代入

**インスタンス** : circle1

# クラス まとめ 例

Circles

`pi = 3.141592653589793`

`__init__`メソッド (初期化)

`circle1.radius = 2`

areaメソッド

`def area(self):`

`return circle1.pi * circle1.radius ** 2`

`circle1 = Circles (2)`

インスタンスを生成

クラス : Circles

クラス変数 : 円周率pi (どの円を作成するときにも共通する値)

インスタンス変数 : 半径self.radius (作成する円によって値が変わる)

`__init__`メソッド (初期化) : インスタンス生成時、selfにcircle1、rに2を代入

インスタンス : circle1

## 練習問題③

---

クラスTriangleを作成

Triangle内に三角形の面積を求めるメソッドを作成

底辺 = 6、高さ = 10 の三角形triangle1

底辺 = 2、高さ = 5 の三角形triangle2      を作成

learn\_python/kadai/day3内に【2019\_12\_04【自分の名前】.py】を

pull request する

pull requestしたらissue【2019/12/04 課題】にコメント