

クラス①

2019/12/04

木南 貴志

オブジェクト指向

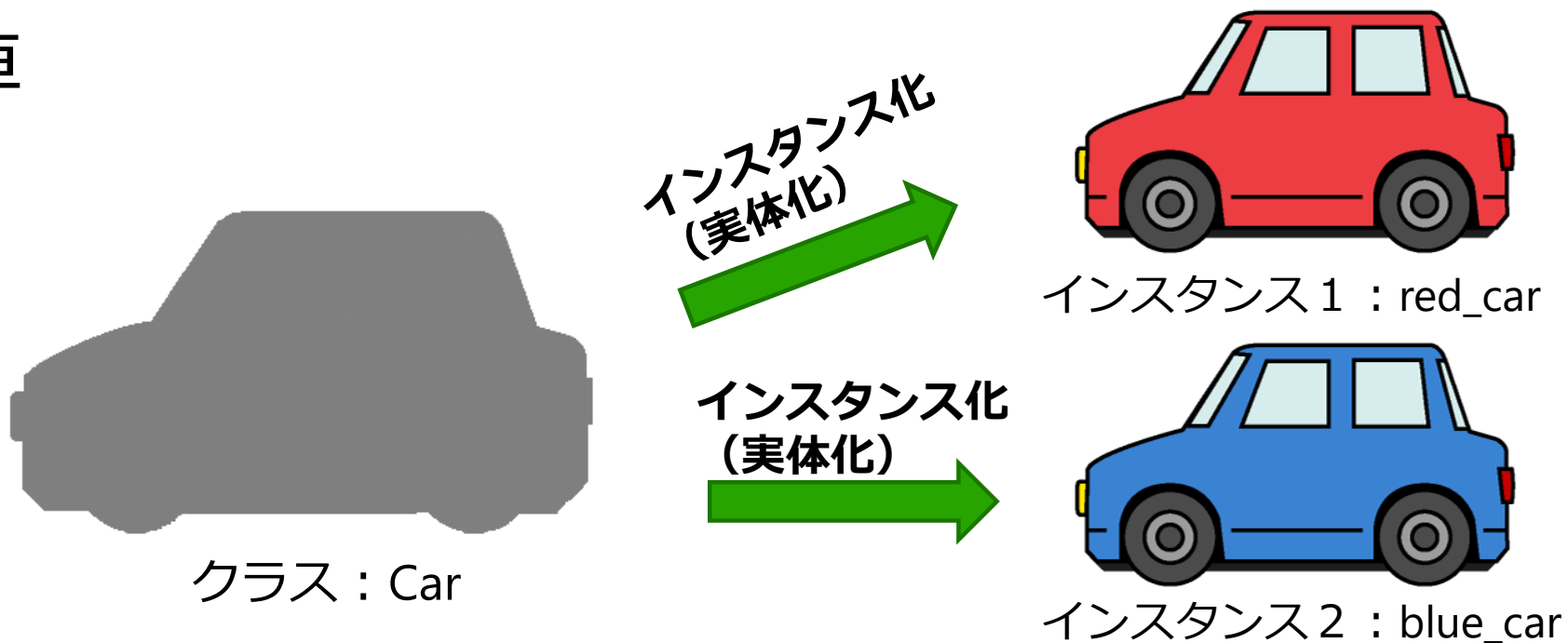
Pythonはオブジェクト指向の言語

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

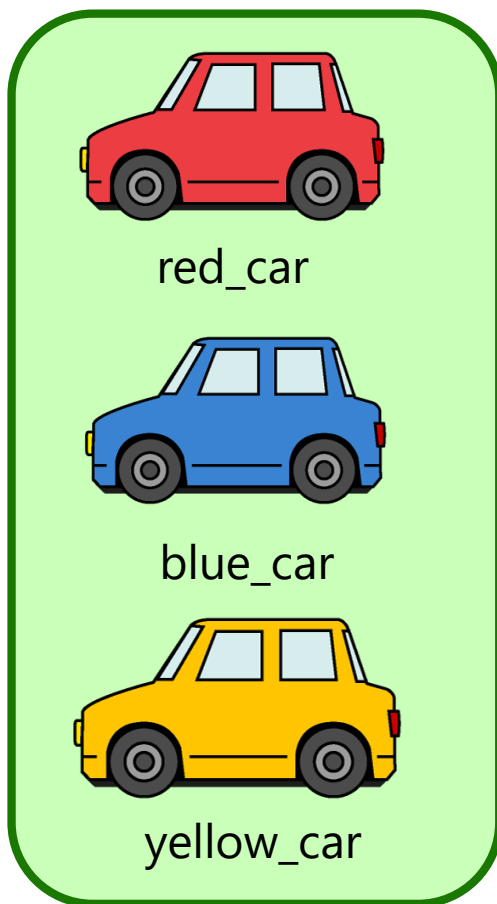
インスタンス：クラス（型）からインスタンスを生成（実体化）

例）車



オブジェクト指向をする理由①

•作業量を省略可能

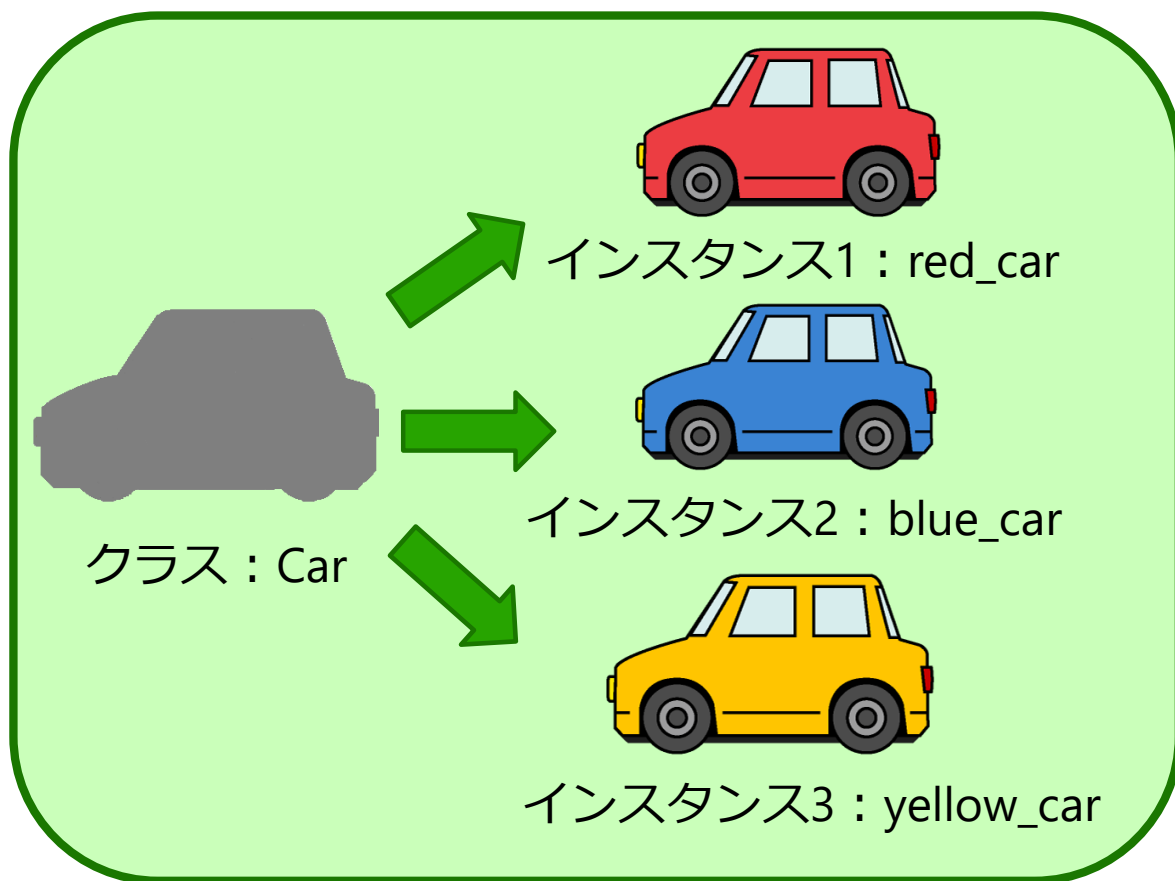


クラスを使用しない場合

- **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

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

クラスのコードの例

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

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通例

クラスのコードの例

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

- **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

```
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

```
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

クラスのコードの例

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**

※メソッドについて次ページで詳しく説明

メソッド

クラス内に作成された関数を**メソッド**という

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

結果

run

- ①クラス内でメソッドを作成
 - ②インスタンスを作成
 - ③インスタンス名.メソッド名
でインスタンスのメソッドを実行
- ※左の場合 red_car.car_run で実行

補足 データ型のクラスとメソッド

通常の変数もクラスに属している

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

結果

```
<class '__main__.Car'>
```

red_carはCarというクラスに属している
(今回は__main__は無視)

```
a = [1,2,3,4]
print(type(a))
```

結果

```
<class 'list'>
```

aはlistというクラスに属している

補足 データ型のクラスとメソッド

通常の変数もクラスに属しているためメソッドが存在

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

結果

run

Carのメソッドcar_runを実行

```
a = [1, 2, 3, 4]
a.append(5)
print(a)
```

結果

[1, 2, 3, 4, 5]

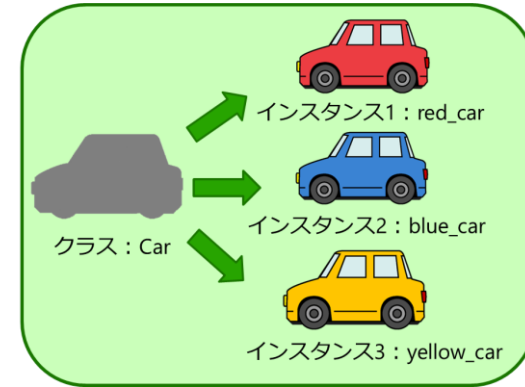
listのメソッドappendを実行

クラスのコードの例

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

__init__ (コンストラクタ) で

self.color = "red" (つまり red_car.color = red)

を実行

色 (color) が赤 ("red") の車 (red_car) を生成

オブジェクト指向をする理由②

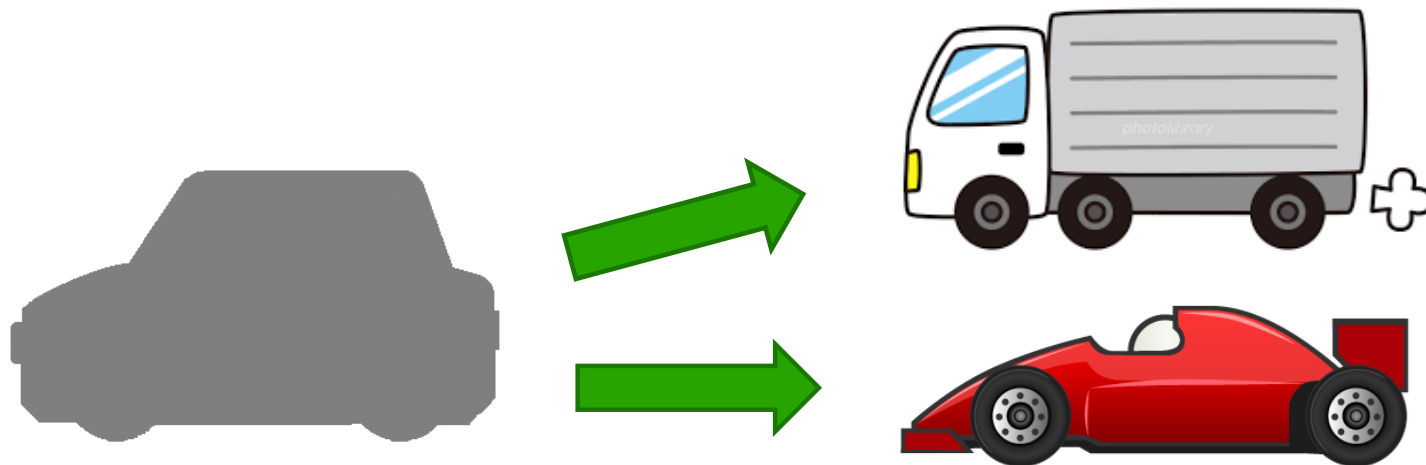
大人数で開発するときに便利（車を例に説明）

- ・「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

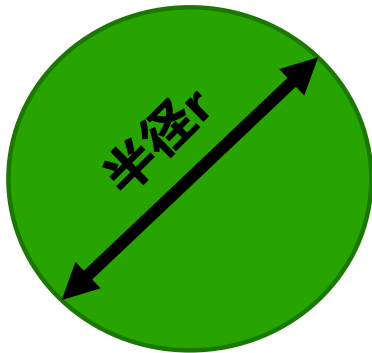
上手く再利用すれば作業効率が向上

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

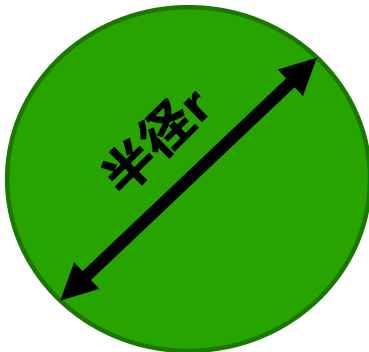


- ・クラス変数（全ての円が共通した値をもつ変数）：
円周率（ $=3.141592653589793$ ）
- ・インスタンス変数（円によって異なる値を持つ変数）：
半径

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
をクラス変数として宣言

self.radius = r
をインスタンス変数として宣言

クラス変数、インスタンス変数

クラス変数、インスタンス変数、メソッド内の変数の比較

クラス変数の参照

```
class Circles:  
    pi = 3.141592653589793
```

```
def __init__(self, r):  
    self.radius = r  
    hoge = 1
```

```
circle_a = Circles(2)  
print(circle_a.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
```

```
circle_a = Circles(2)  
print(circle_a.radius)
```

結果

2

メソッド内の変数の参照

```
class Circles:  
    pi = 3.141592653589793
```

```
def __init__(self, r):  
    self.radius = r  
    hoge = 1
```

```
circle_a = Circles(2)  
print(circle_a.hoge)
```

結果（エラーが発生）

'Circles' object has no attribute 'hoge'

変数の種類によって特徴が違う

練習問題

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