クラス①

2019/12/04

木南　貴志

オブジェクト指向

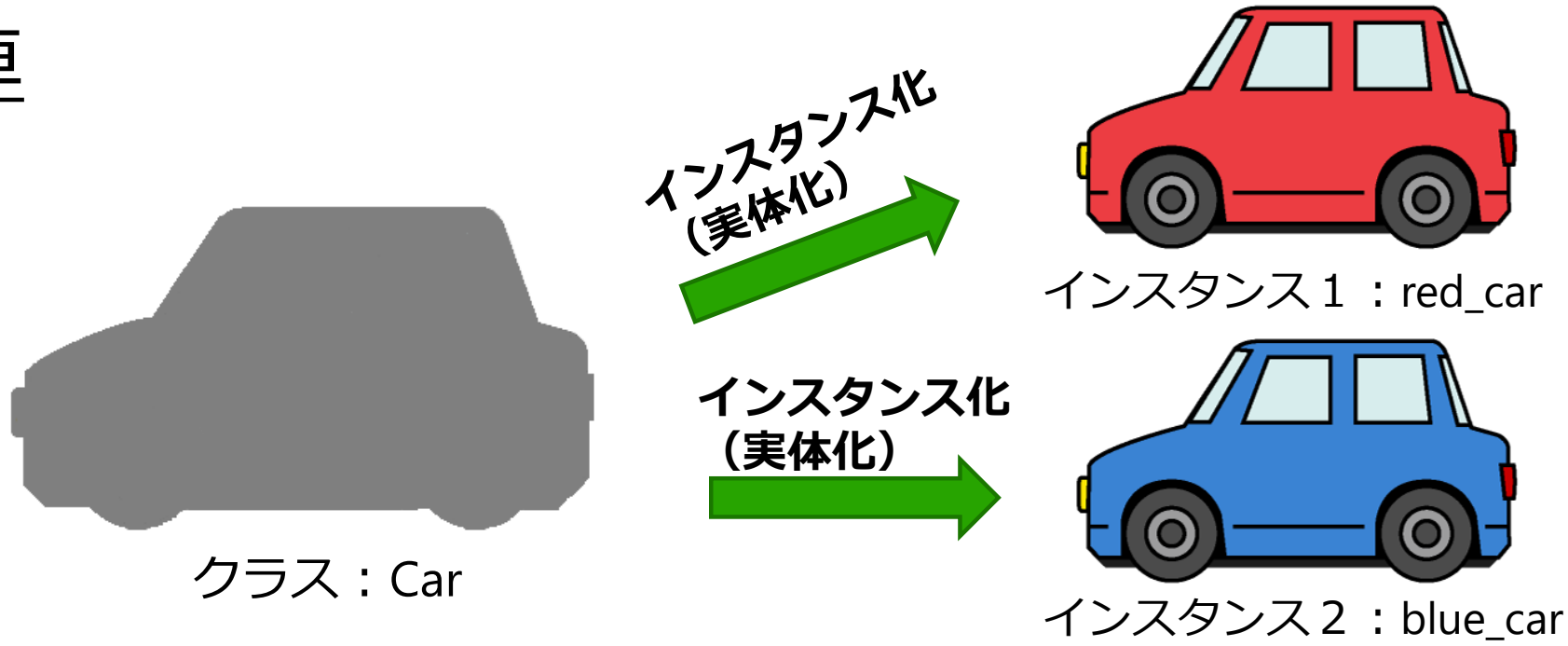
Pythonはオブジェクト指向の言語

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


blue_car


yellow_car

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

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

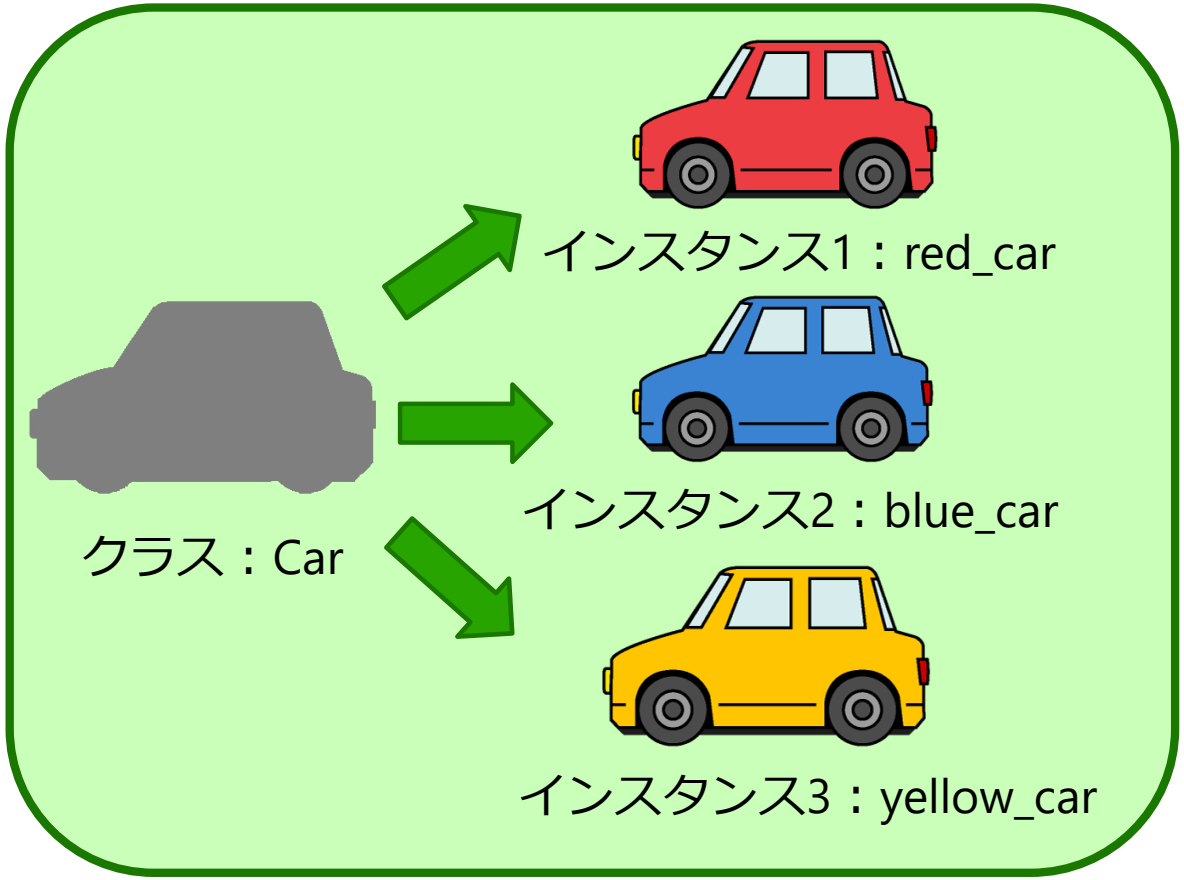
yellow_carで
「走る」「止まる」
機能を設定

同じ設定を
車ごとに行う必要あり

https://eng-entrance.com/what-oop

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
```

・クラスを使用する場合

車が共通して持つ機能はクラス内で記述

```
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

・**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）

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

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

一致

red_carを生成する場合
self に red_car
color に “red” が代入

結果

```
red
```

selfを指定ない場合（ローカル変数）

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

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

不一致

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'
```

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

__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）

③car_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）

③car_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()
```

car_runメソッドを呼び出す

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）

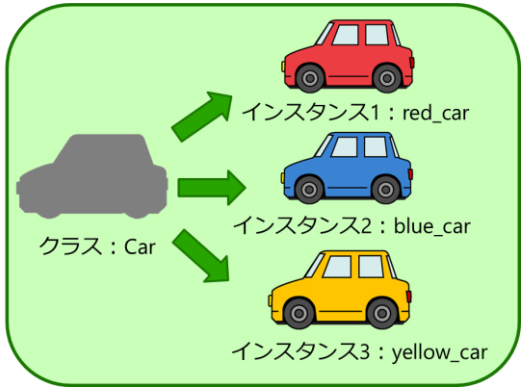
③car_runメソッドを呼び出し実行

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**

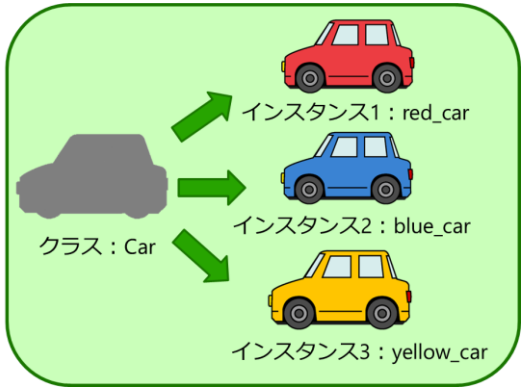
  インスタンス作成時は必ず

  __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

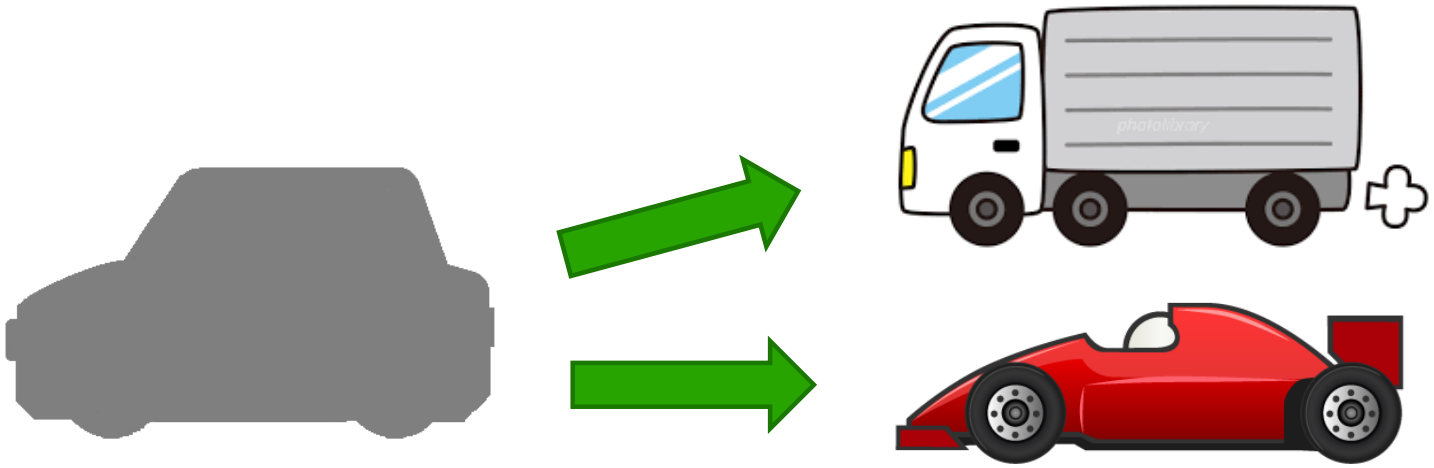
・「車」を用意しない
➡「走る」「止まる」のコードを理解する必要あり
➡正しく理解しないと「走る」「止まる」の部分を破壊する恐れあり

「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

https://eng-entrance.com/what-oop

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）



レージングカー、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

https://eng-entrance.com/what-oop

オブジェクト指向をする理由③

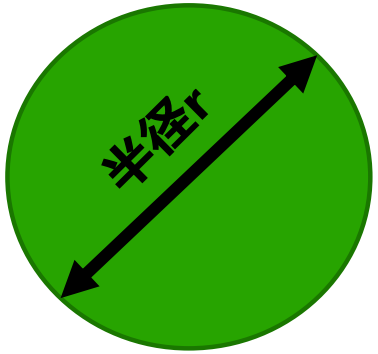
同じようなモノを作りやすい（具体例）

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)

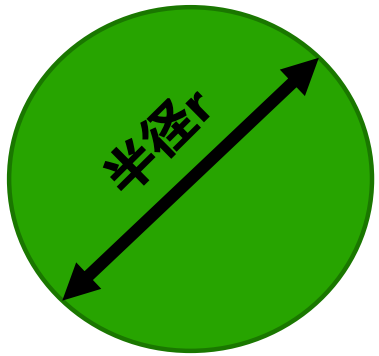
・インスタンス変数（円によって異なる値を持つ変数）：
半径

https://python.atelierkobato.com/instance/

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

https://python.atelierkobato.com/instance/

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

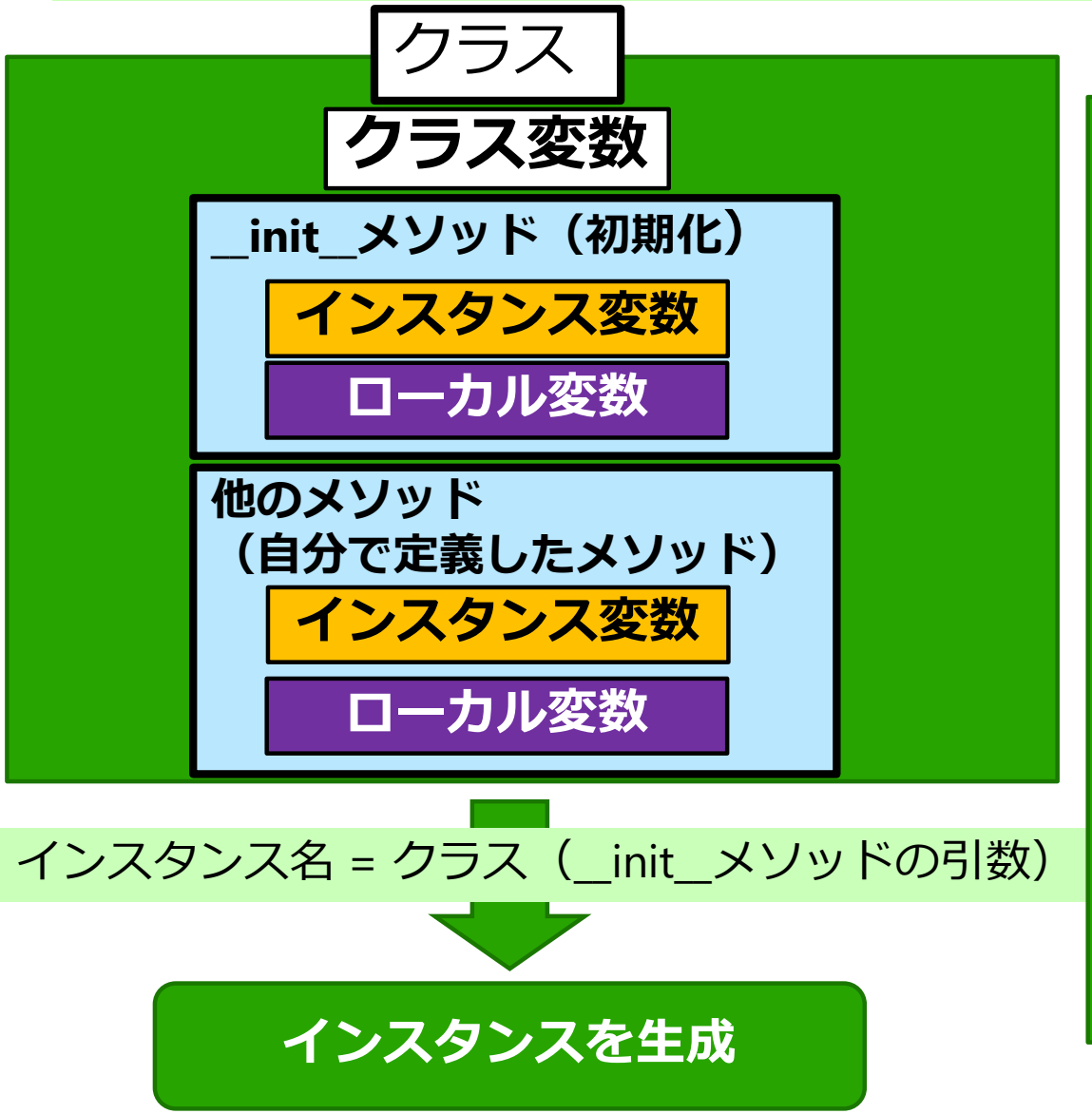
```
class Circles:
    pi = 3.141592653589793

    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

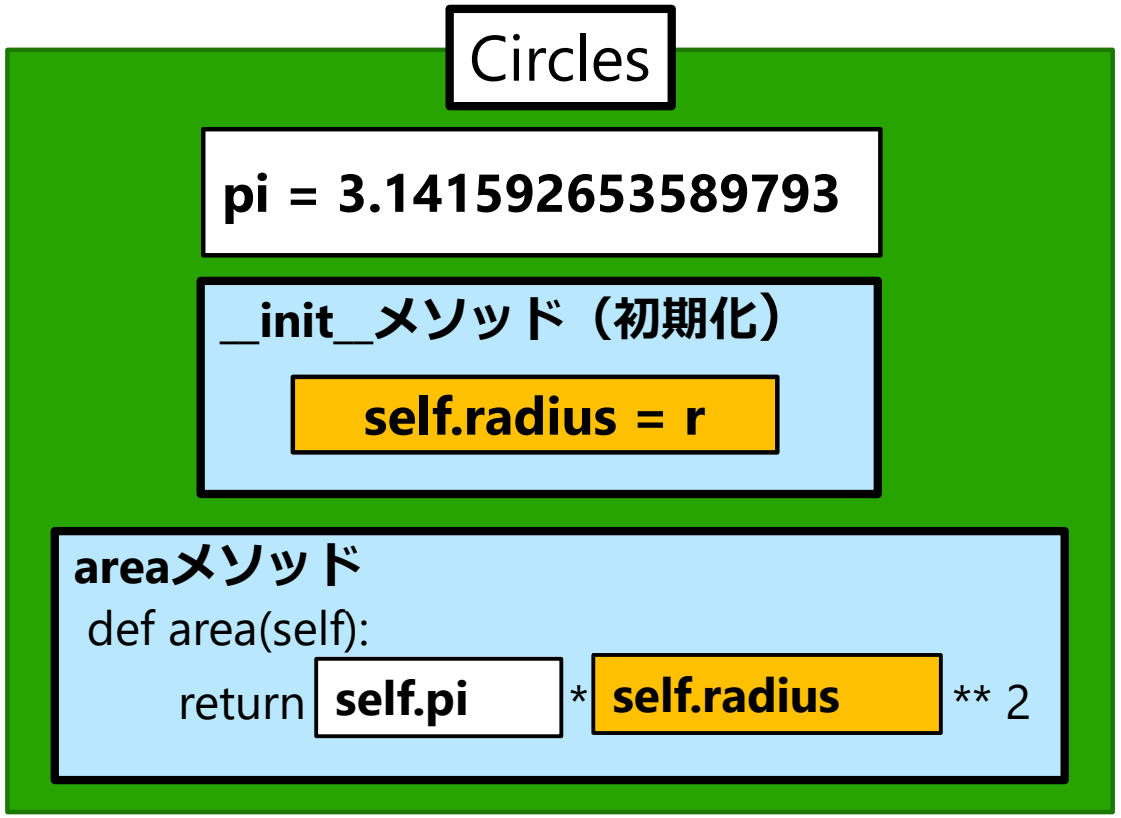
クラス変数：どのインスタンスを生成する時も共通する値をもつ

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



クラス：Circles

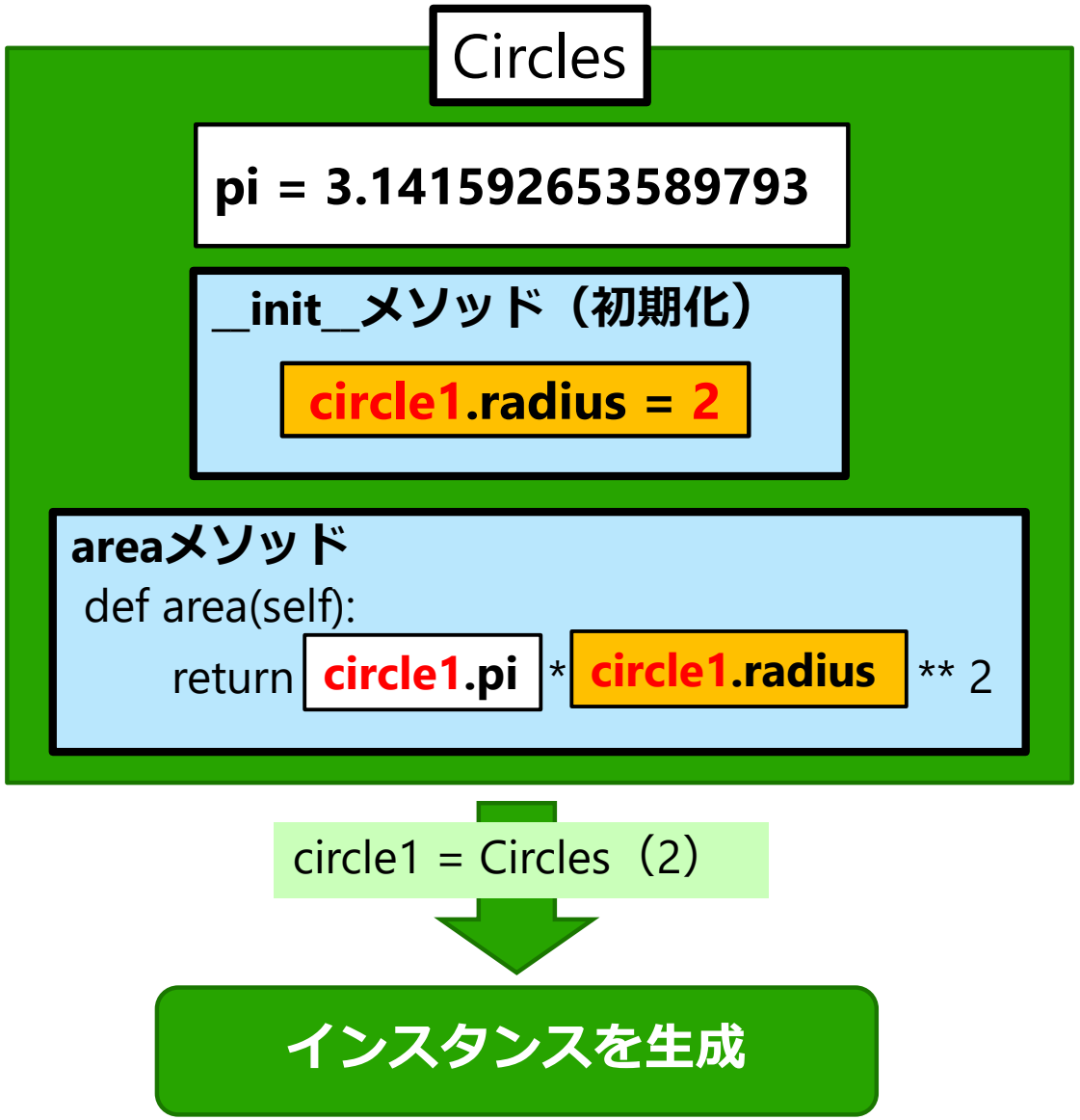
クラス変数：円周率pi（どの円を作成するときにも共通する値）

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



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