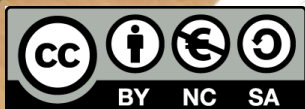


po-9. Python プログラ ミング演習の総まとめ

プログラミングの基礎を
Python で学びたい人へ (全9回)

URL: <https://www.kkaneko.jp/cc/po/index.html>

金子邦彦



アウトライン

9-1 オブジェクト, メソッド, 関数

9-2 クラス定義, コンストラクタ

9-3 クラス階層, 継承

- 重要トピックのまとめ

9-1. オブジェクト, メソッド, 関数

オブジェクト

- プログラミングでのオブジェクトは、コンピュータでの操作や処理の対象となるもののこと。
- オブジェクトは1つあるいは複数のデータを持つことができる

メソッド

- **メソッド**は、オブジェクトに属する操作や処理のこと

hero.moveDown()

hero は**オブジェクト**

moveDown() は**メソッド**

間を「.」で区切っている

- **引数（ひきすう）**とは、メソッドに渡す値のこと

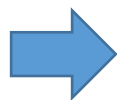
式の抽象化



$$100 * 1.1$$

$$150 * 1.1$$

$$400 * 1.1$$



$$a * 1.1$$

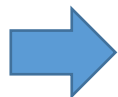
変数 a を使って, 複数の**式**を1つにまとめる
(**抽象化**)

類似した複数の**式**

関数



100 * 1.1



a * 1.1

150 * 1.1

変数 a を使って, 複数の式を1つにまとめる
(抽象化)

400 * 1.1

類似した複数の式



```
1 def foo(a):  
2     return a * 1.1  
3 print(foo(100))  
4 print(foo(150))  
5 print(foo(400))
```

式「a * 1.1」を含む
関数 foo を定義

関数 foo を使用.
100, 150, 400 は引数

Print output (drag lower right)

```
110.000000000000000001  
165.0  
440.00000000000000006
```

```
public static double foo(double a) {  
    return a * 1.1;  
}
```

- この**関数の本体**は「`return a * 1.1;`」
- この**関数**は、式「`a * 1.1`」に、名前 `foo` を付けたものと考えることもできる

式の抽象化と関数



抽象化前

```
1 print(100 * 1.1)
2 print(150 * 1.1)
3 print(400 * 1.1)
```

類似した複数の式

Print output (drag lower right)

```
110.000000000000000001
165.0
440.000000000000000006
```

実行結果

抽象化後

```
1 def foo(a):
2     return a * 1.1
3 print(foo(100))
4 print(foo(150))
5 print(foo(400))
```

関数の定義と使用

Print output (drag lower right)

```
110.000000000000000001
165.0
440.000000000000000006
```

同じ
実行結果になる

まとめ



- プログラミングでの**オブジェクト**は、コンピュータでの操作や処理の対象となるもののこと
- **メソッド**は、オブジェクトに属する操作や処理のこと
- 次の**関数**は、式「a * 1.1」に、名前 foo を付けたものと考えることもできる

```
public static double foo(double a) {  
    return a * 1.1;  
}
```

- **式の抽象化**とは、**変数**を使って、複数の**式**を1つにまとめること

9-2. クラス定義, コンストラクタ

Python のオブジェクトの生成



- 次の2つの**オブジェクト**を生成する Python プログラム

a	1	2	"red"
b	2	4	"green"
	x	y	color

- クラス Ball のオブジェクト生成を行うプログラム

```
a = Ball(1, 2, "red")  
b = Ball(2, 4, "green")
```

クラス定義



クラス定義の中には、**コンストラクタ**の定義、その他**メソッド**の定義を含める。

```
1 class Ball:
2     def __init__(self, x, y, color):
3         self.x = x
4         self.y = y
5         self.color = color
6     def move(self, xx, yy):
7         self.x = self.x + xx
8         self.y = self.y + yy
9     def right(self, a):
10        self.move(a, 0)
11    def left(self, a):
12        self.move(-a, 0)
```

コンストラクタ

その他
メソッド

オブジェクトの生成を行う**メソッド**のことを
コンストラクタという

メソッドと クラス

- プログラミングでの**メソッド**とは、オブジェクトに関する操作や処理のこと
- **メソッド**は、**クラス**に属する
- **メソッド**内のプログラムは、その**メソッド**が所属する**クラス**の**属性**や**メソッド**へのアクセス権がある

属性やメソッド のアクセス

- 「オブジェクト名」 + 「.」で
属性やメソッドにアクセスす
る

属性アクセス



x	5	170.51	"apple"
---	---	--------	---------

qty weight name

「オブジェクト名」 + 「.」で属性やメソッドにアクセスする

```
1 class C:
2     def __init__(self, qty, weight, name):
3         self.qty = qty
4         self.weight = weight
5         self.name = name
6     def total(self):
7         return self.qty * self.weight
8
9 x = C(5, 107.51, "apple")
10 y = C(3, 40.97, "orange")
11 print(x.qty)
12 print(x.total())
```


メソッドアクセス



「オブジェクト名」 + 「.」で属性やメソッドにアクセスする。

メソッド内でアクセスするときも「.」

※ 下のプログラムで、「**self**」は、「メソッドが処理中のオブジェクト」を扱うための変数

```
1 class C:
2     def __init__(self, qty, weight, name):
3         self.qty = qty
4         self.weight = weight
5         self.name = name
6     def total(self):
7         return self.qty * self.weight
8
9 x = C(5, 107.51, "apple")
10 y = C(3, 40.97, "orange")
11 print(x.qty)
12 print(x.total())
```

まとめ



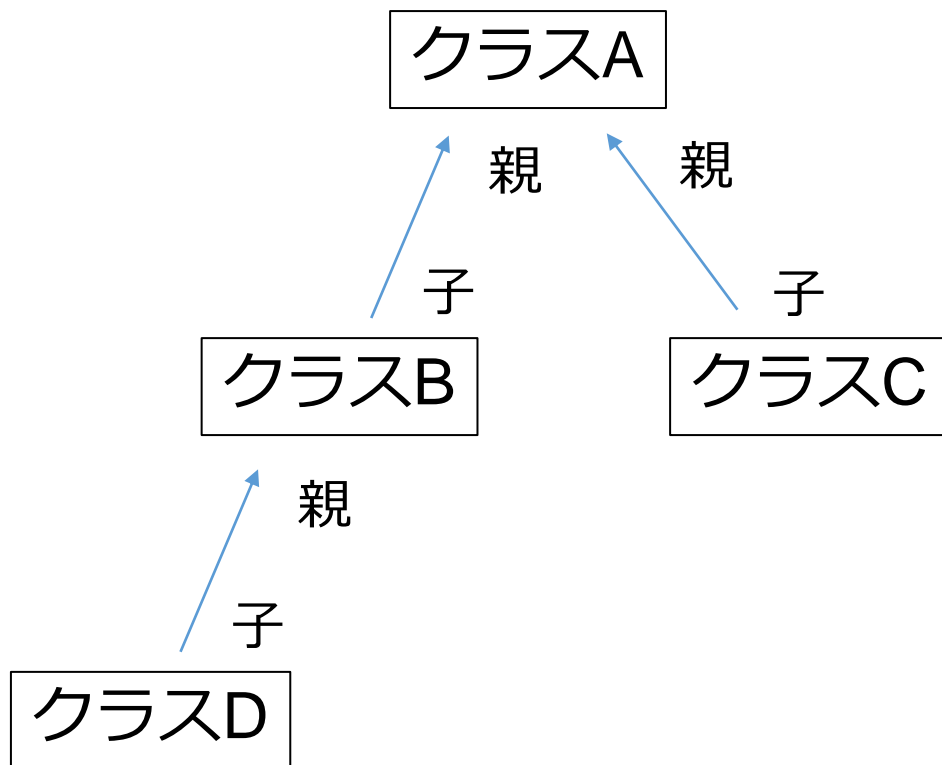
- **メソッド**は, **クラス**に属する
- **「オブジェクト名」 + 「.」**で**属性**や**メソッド**に**アクセス**する

9-3. クラス階層, 継承

クラス階層



クラス階層とは、複数のクラスが親子関係をなすこと



- **継承**とは、**親クラスの属性とメソッド**を**子クラス**が**受け継ぐ**こと
- **親クラス**のことを「**スーパークラス**」、**子クラス**のことを「**サブクラス**」ともいう

Python のオブジェクトの生成



- 次の3つの**オブジェクト**を生成

a	1	2	"red"	
b	3	4	"green"	
x	5	6	"blue"	1
	x	y	color	r

- オブジェクト生成を行うプログラム

```
Ball a = new Ball(1, 2, "red");  
Ball b = new Ball(2, 4, "green");  
Circle x = new Circle(5, 6, "blue", 1);
```

クラスの類似性



- 類似した2つの**クラス**

Ball

属性

x
y
color

メソッド

move
reset

Circle

属性

x
y
color
r

move
reset



x, y, color は同じ

r の有り無しが
違う



メソッドの名前も
中身も全く同じとする

2つのクラスのプログラム (親子関係にしない場合)



Ball

```
class Ball:
```

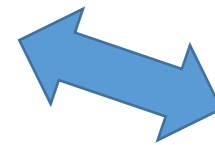
```
def __init__(self, x, y, color):  
    self.x = x  
    self.y = y  
    self.color = color
```

```
def move(self, xx, yy):  
    self.x = self.x + xx  
    self.y = self.y + yy
```

```
def reset(self):  
    self.x = 0  
    self.y = 0
```



追加



全く同じ

Circle

```
class Circle:
```

```
def __init__(self, x, y, color, r):  
    self.x = x  
    self.y = y  
    self.color = color  
    self.r = r
```

```
def move(self, xx, yy):  
    self.x = self.x + xx  
    self.y = self.y + yy
```

```
def reset(self):  
    self.x = 0  
    self.y = 0
```


問いかけ



同じようなプログラムを繰り返し書きたいですか？

-> **No. クラス階層により解決**

Ball

Circle

```
class Ball:
```

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

```
def move(self, xx, yy):  
    self.x = self.x + xx  
    self.y = self.y + yy
```

```
def reset(self):  
    self.x = 0  
    self.y = 0
```



追加

```
class Circle:
```

```
def __init__(self, x, y, color, r):  
    self.x = x  
    self.y = y  
    self.color = color  
    self.r = r
```

```
def move(self, xx, yy):  
    self.x = self.x + xx  
    self.y = self.y + yy
```

```
def reset(self):  
    self.x = 0  
    self.y = 0
```

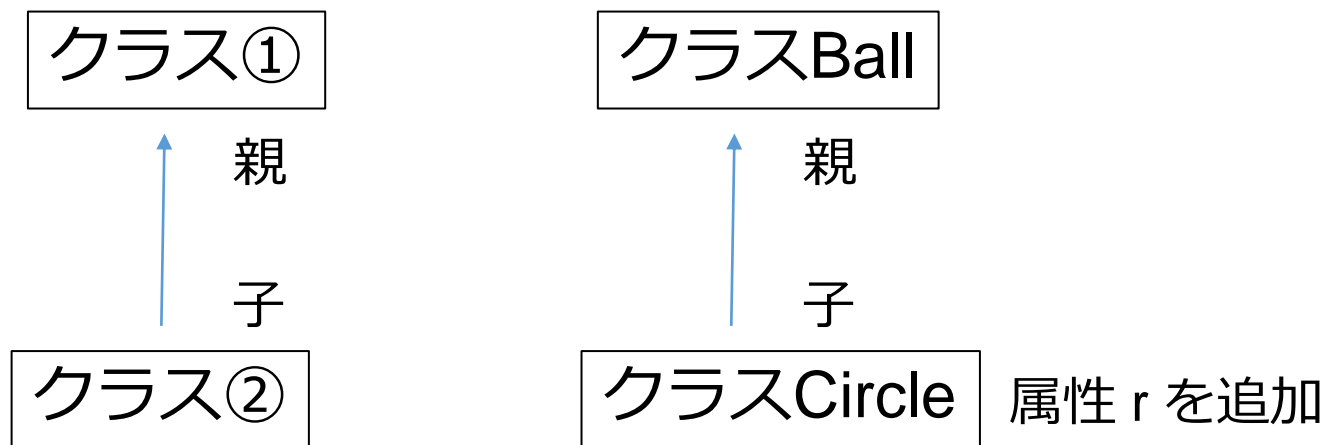


全く同じ

クラスの親子関係



- クラス①が**親**，クラス②が**子**であるとき
 - クラス②は，クラス①の**属性**と**メソッド**を**すべて持つ**
 - クラス②で，クラス①にない**属性**や**メソッド**が**追加される**ことがある



クラスの親子関係



```
class Ball:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
    def move(self, xx, yy):
        self.x = self.x + xx
        self.y = self.y + yy
    def reset(self):
        self.x = 0
        self.y = 0
```

クラス名 **Ball**

属性 **x, y, color**

メソッド **move, reset**

```
class Circle(Ball):
    def __init__(self, x, y, color, r):
        super(Circle, self).__init__(x, y, color)
        self.r = r
```

クラス名 **Circle**

属性 **x, y, color, r**

メソッド **move, reset**

クラス **Circle** は、**親クラス**であるクラス **Ball** の**属性**と**メソッド**を**継承**する。

2つのクラスのプログラム 親子関係にしない場合とする場合の比較



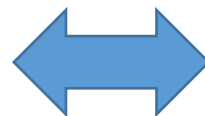
Ball

```
class Ball:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
    def move(self, xx, yy):
        self.x = self.x + xx
        self.y = self.y + yy
    def reset(self):
        self.x = 0
        self.y = 0
```

Ball

```
class Ball:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
    def move(self, xx, yy):
        self.x = self.x + xx
        self.y = self.y + yy
    def reset(self):
        self.x = 0
        self.y = 0
```

働きは
同じ



Circle

```
class Circle {
    double x;
    double y;
    String color;
    double r;
    public Circle(double x, double y, String color, double r) {
        this.x = x;
        this.y = y;
        this.color = color;
        this.r = r;
    }
    public void move(double xx, double yy) {
        this.x = this.x + xx;
        this.y = this.y + yy;
    }
    public void reset() {
        this.x = x;
        this.y = y;
    }
}
```

Circle

```
class Circle(Ball):
    def __init__(self, x, y, color, r):
        super(Circle, self).__init__(x, y, color)
        self.r = r
```

親子関係にしない

(同じようなプログラムを繰り返す)

親子関係にする

Python でのクラスの親子関係の書き方



親子関係の指定 「**class Circle(Ball):**」

子クラスである Circle で追加される
属性, メソッドを書く

```
class Circle(Ball):  
    def __init__(self, x, y, color, r):  
        super(Circle, self).__init__(x, y, color)  
        self.r = r
```

コンストラクタの定義.

`super(Circle, self).__init__` により, 親クラスの
コンストラクタを呼び出していることに注意

Python でのクラスの親子関係の書き方



```
class Circle(Ball):
```

```
    def __init__(self, x, y, color, r):  
        super(Circle, self).__init__(x, y, color)  
        self.r = r
```

親クラスの**コンストラクタ**を
呼び出して、**属性値**を設定

子クラスで追加した**属性**を設定

まとめ



- **クラス階層**とは、複数のクラスが親子関係をなすこと
- クラス①が**親**，クラス②が**子**であるとき
 - クラス②は，クラス①の**属性とメソッド**をすべて持つ
 - クラス②で，クラス①にない**属性やメソッド**が追加されることがある
- **親子関係**の指定は，「**class Circle(Ball):**」のよ
うに書く． Circle が子， Ball が親．
- **継承**とは，**親クラス**の**属性とメソッド**を**子クラス**
が受け継ぐこと
- **親クラス**のことを「**スーパークラス**」， **子クラス**
のことを「**サブクラス**」ともいう

関連資料

- Python まとめページ

<https://www.kkaneko.jp/cc/python/googlecolab.html>

- Python の基本

Google Colaboratory, Paiza.IO を使用.

<https://www.kkaneko.jp/cc/colab/index.html>

- Python 入門 (全6回)

Google Colaboratoryを使用.

<https://www.kkaneko.jp/cc/pf/index.html>

- Python プログラミング演習 (全9回)

Python Tutor, VisuAlgo を使用

<https://www.kkaneko.jp/cc/po/index.html>

- さまざまな Windows アプリケーションのインストールと設定

<https://www.kkaneko.jp/cc/tools/index.html>