

pf-11. クラス階層, 繙承

(Python 入門)

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

金子邦彦



オブジェクトとメソッド



- ・**オブジェクト**：コンピュータでの操作や処理の対象となるもの

hero.moveDown()

hero オブジェクト
moveDown() メソッド
間を「.」で区切っている

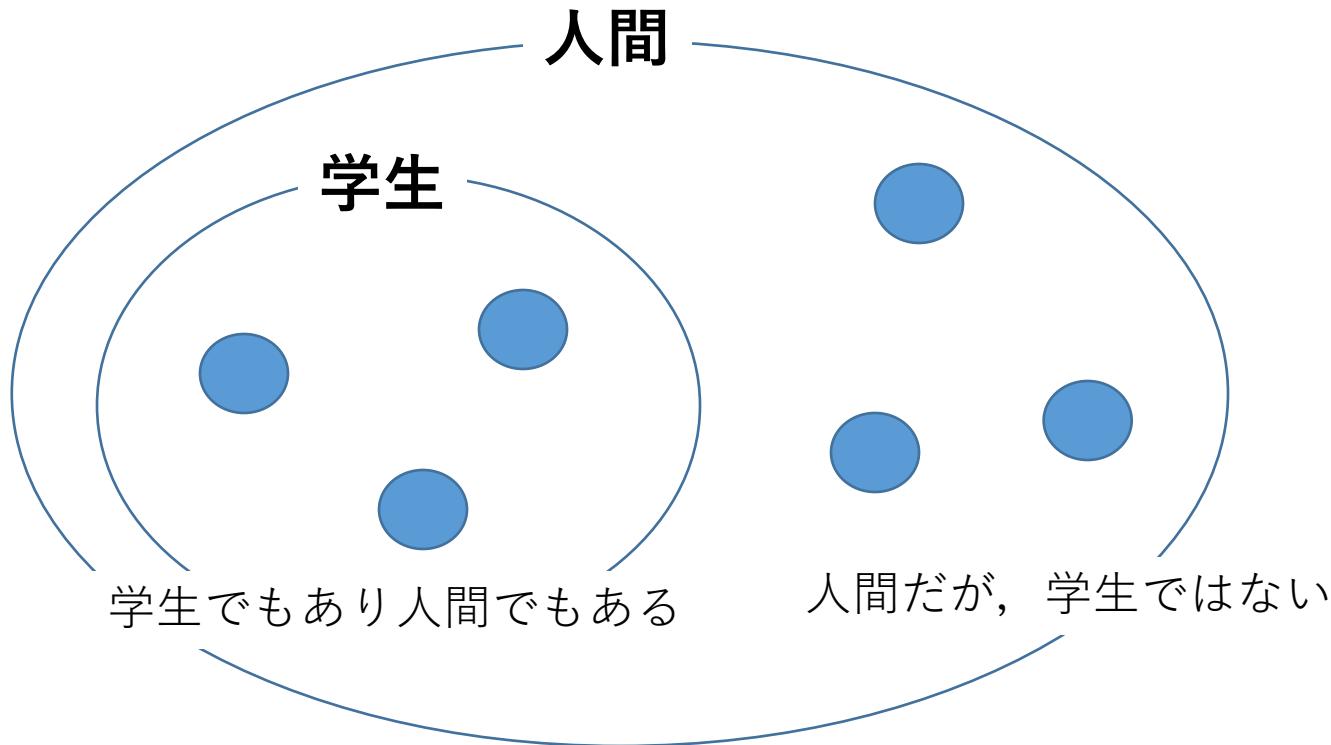
- ・**メソッド**：オブジェクトに属する機能や操作。オブジェクトがもつ能力に相当する
- ・**引数**：メソッドが行う操作の詳細に関する情報，メソッド呼び出しのときに、引数を指定できる

hero.attack("fence", 36, 26)

クラスとオブジェクト



クラスは、同じ種類のオブジェクトの集まりと考え
ることができる



クラス定義の例

```
1 class Point:  
2     def __init__(self, x, y, color):  
3         self.x = x  
4         self.y = y  
5         self.color = color  
6     def printout(self):  
7         print(self.x, self.y, self.color)  
8  
9 p = Point(1, 2, "red")  
10 p.printout()
```

クラス名: Point

属性: x, y, color

メソッド: __init__, printout

オブジェクト生成の際に、メソッド __init__ が自動で実行される

- Trinket はオンラインの Python、HTML 等の学習サイト
- 有料の機能と無料の機能がある
- 自分が作成した Python プログラムを公開し、他の人に実行してもらうことが可能（そのとき、書き替えて実行也可能）
- Python の標準機能を登載、その他、次のモジュールやパッケージがインストール済み

math, matplotlib.pyplot, numpy, operator, processing, pygal,
random, re, string, time, turtle, urllib.request



trinket でのプログラム実行

- trinket は Python, HTML などのプログラムを書き実行できるサイト
- <https://trinket.io/python/0fd59392c8>

のように、違うプログラムには違う URL が割り当てられる

The screenshot shows the trinket.io web interface. At the top, there's a navigation bar with a home icon, 'My Trinkets', and a file name 's11-1'. Below the bar is the trinket logo and a 'Run' button, which is highlighted with a red box. To the left is a code editor window titled 'main.py' containing the following Python code:

```
1 age = 18
2 if age <= 11:
3     print(500)
4 else:
5     print(1800)
```

To the right of the code editor is a 'Result' panel where the output of the program will be displayed.

ソースコードの
メイン画面

実行結果

- 実行が開始しないときは、「実行ボタン」で実行
- ソースコードを書き替えて再度実行することも可能

演習

資料：8

【トピックス】

- クラス定義
- class
- オブジェクト生成
- メソッド呼び出し

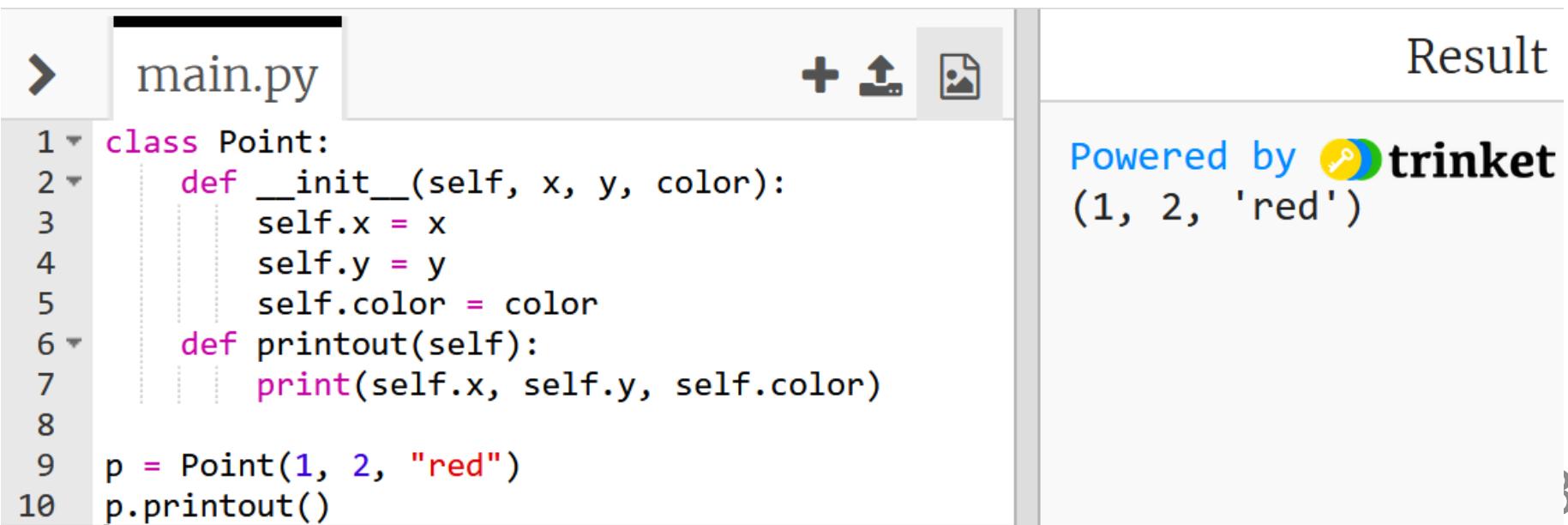


① trinket の次のページを開く

<https://trinket.io/python/b0698edcb7>

② 実行結果が、次のように表示されることを確認

このプログラムは、**オブジェクト p を生成する**. そして、
メソッド printout を呼び出して, 属性値を表示させる



The screenshot shows the trinket web-based Python editor interface. On the left, the code editor window titled "main.py" contains the following Python script:

```
1 class Point:
2     def __init__(self, x, y, color):
3         self.x = x
4         self.y = y
5         self.color = color
6     def printout(self):
7         print(self.x, self.y, self.color)
8
9 p = Point(1, 2, "red")
10 p.printout()
```

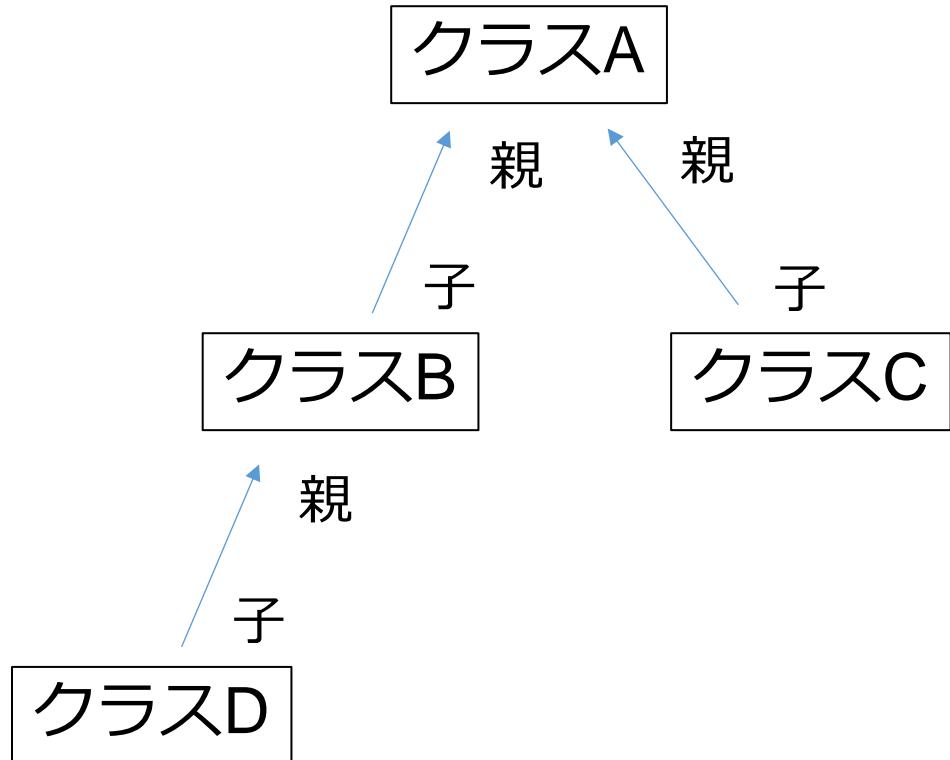
On the right, the "Result" panel displays the output of the program's execution:

Powered by  trinket
(1, 2, 'red')

クラス階層



クラス階層では、複数のクラスが親子関係をなす



クラス Point

オブジェクト



場所 (1, 2)
色 red

クラス Point

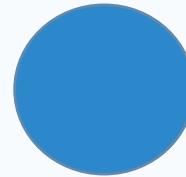


クラス Ball

クラス Ball

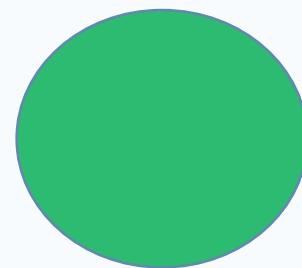


オブジェクト



半径 1, 場所 (8, 10)
色 blue

オブジェクト



半径 3, 場所 (2, 4)
色 green

類似した 2つのクラス



Point

属性

x
y
color

メソッド

printout

↔
x, y, color は同じ
r の有り無しが
違う

Ball

属性

x
y
color
r

メソッド

printout

printout は名前は
同じだが、中身が違う

継承



- 親クラスの属性とメソッドは、子クラスに継承される
- 子クラスにおいて、同じ名前のメソッドが別定義されることもある

親

Point

```
class Point:  
    def __init__(self, x, y, color):  
        self.x = x  
        self.y = y  
        self.color = color  
    def printout(self):  
        print(self.x, self.y, self.color)
```

子

Ball

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

- 属性 r を追加
- メソッド printout は別定義

属性 r を追加

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

メソッド printout は別定義

Point クラスと Ball クラスの定義の例 (クラス階層を考えない場合)



Point

```
class Point:  
    def __init__(self, x, y, color):  
        self.x = x  
        self.y = y  
        self.color = color  
    def printout(self):  
        print(self.x, self.y, self.color)
```

Ball

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

x, y, color は同じ

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

Point クラスと Ball クラスの定義の例 (クラス階層を考える場合)



Point

```
class Point:  
    def __init__(self, x, y, color):  
        self.x = x  
        self.y = y  
        self.color = color  
    def printout(self):  
        print(self.x, self.y, self.color)
```

Ball

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

x, y, color について
繰り返し書くことはなくなる

class Ball(Point)

Ball クラスは Point クラスの子である

Point

```
class Point:
    def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
    def printout(self):
        print(self.x, self.y, self.color)
```

Ball

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

super(Ball, self).__init__(x, y, color)

親クラスである Point クラスの
メソッド __init__ にアクセス。
その引数は x, y, color

クラス階層を考える場合と考えない場合の違い



Point

```
class Point:  
    def __init__(self, x, y, color):  
        self.x = x  
        self.y = y  
        self.color = color  
    def printout(self):  
        print(self.x, self.y, self.color)
```

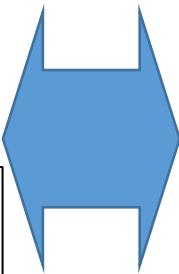
動きは
同じ

Point

```
class Point:  
    def __init__(self, x, y, color):  
        self.x = x  
        self.y = y  
        self.color = color  
    def printout(self):  
        print(self.x, self.y, self.color)
```

Ball

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



Ball

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

クラス階層を考えない

クラス階層を考える



演習

資料：19

【トピックス】

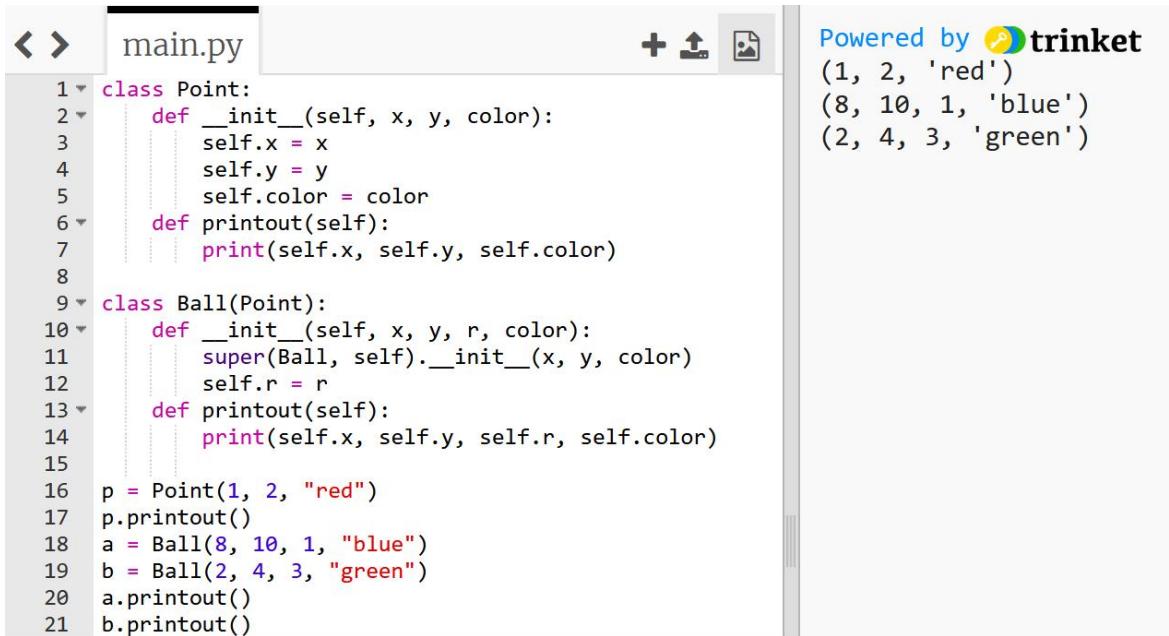
- クラス定義
- Class
- サブクラスのクラス定義

① trinket の次のページを開く

<https://trinket.io/python/8f5efe7d0e>

② 実行結果が、次のように表示されることを確認

このプログラムは、オブジェクト p, a, b を生成する。そして、
メソッド printout を呼び出して、属性値を表示させる



The screenshot shows a code editor window for 'main.py' on the left and its execution output on the right. The code defines two classes: 'Point' and 'Ball'. The 'Point' class has an __init__ method that initializes x, y, and color attributes. It also has a printout method that prints the values of x, y, and color. The 'Ball' class inherits from 'Point' and adds a radius 'r'. Both classes have a printout method that prints the values of x, y, r, and color. In the main block, objects p, a, and b are created and their printout methods are called. The output shows three tuples: (1, 2, 'red'), (8, 10, 1, 'blue'), and (2, 4, 3, 'green').

```
main.py
Powered by trinket
(1, 2, 'red')
(8, 10, 1, 'blue')
(2, 4, 3, 'green')

1  class Point:
2      def __init__(self, x, y, color):
3          self.x = x
4          self.y = y
5          self.color = color
6      def printout(self):
7          print(self.x, self.y, self.color)
8
9  class Ball(Point):
10     def __init__(self, x, y, r, color):
11         super(Ball, self).__init__(x, y, color)
12         self.r = r
13     def printout(self):
14         print(self.x, self.y, self.r, self.color)
15
16 p = Point(1, 2, "red")
17 p.printout()
18 a = Ball(8, 10, 1, "blue")
19 b = Ball(2, 4, 3, "green")
20 a.printout()
21 b.printout()
```