

po-8. クラス, メソッド, オブジェクト生成

トピックス: クラス, class, コンストラクタ,
メソッド, self

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

(Python プログラミングの基本)

金子邦彦

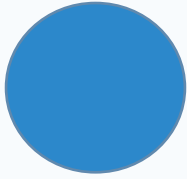


全体まとめ



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

```
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)
```

クラス定義のプログラム

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
```

オブジェクト生成のプログラム

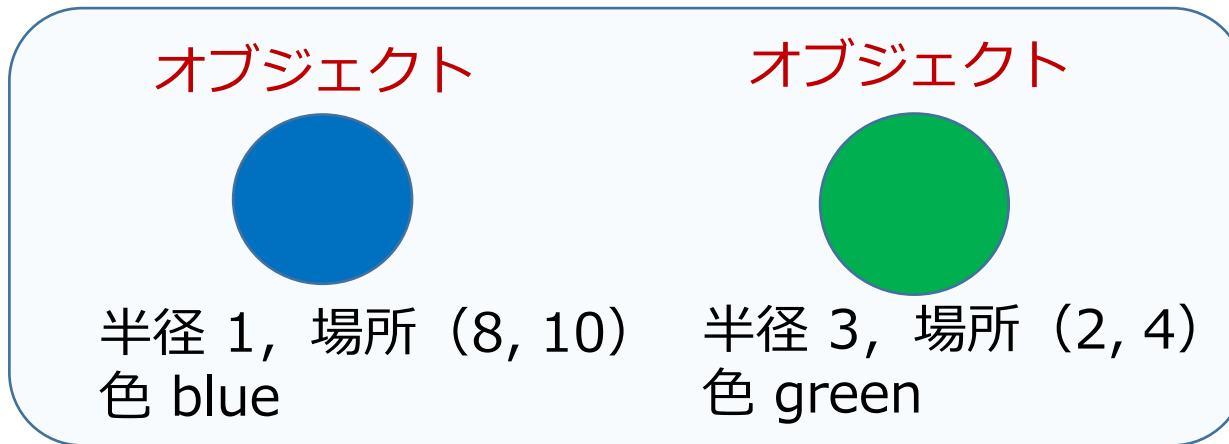
```
a.printout()
b.printout()
```

メソッドアクセスのプログラム

全体まとめ



- 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 Ball

- メソッド定義内で、そのメソッドが所属するクラスで定義された属性やメソッドにアクセスするときは self + 「.」

```
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)
```

	項目
	復習
8-1	クラスとオブジェクト
8-2	クラス定義, class, オブジェクト生成
8-3	メソッドアクセス, 属性アクセス, self
8-4	演習

オブジェクトとメソッド



hero.moveDown() Python プログラム

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**に属する操作や処理.
- **メソッド**呼び出しでは, **引数**を指定することがある. **引数** (ひきすう) は, **メソッド**に渡す値のこと

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

代入



- **代入** : プログラムで, 「**x = 100**」 のように書くと, **x の値が 100 に変化** する

x = 100

プログラム



Frames

Global frame

x | 100

実行結果

Python プログラムの書き方



Python プログラムの例

```
x = 100
a = x + 200
enemy1 = hero.findNearestEnemy()
hero.attack(enemy1)
```

- **代入** : **オブジェクト名** + 「**=**」
+ 式または値またはメソッド呼び出し
- **メソッドアクセス** : **オブジェクト名** + 「**.**」
+ **メソッド名** + 「**()**」 (引数を付けることも)

Python プログラムでは、その他にも、属性アクセス、関数呼び出し、制御、「*」、「+」などの演算子、コマンド、定義など

Python Tutor の起動



① **ウェブブラウザ**を起動する

② **Python Tutor** を使いたいので, 次の URL を開く
<http://www.pythontutor.com/>

③ 「**Python**」 をクリック ⇒ **編集画面**が開く

Learn Python, JavaScript, C, C++, and Java

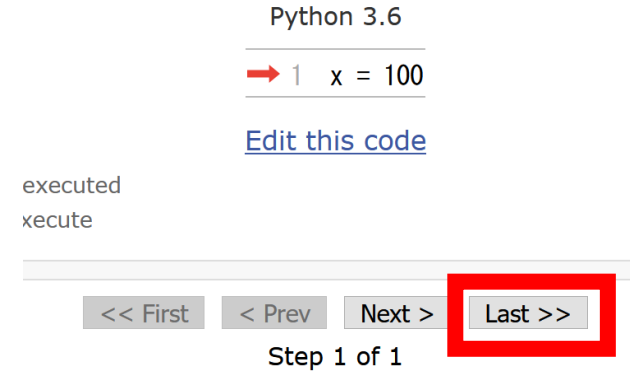
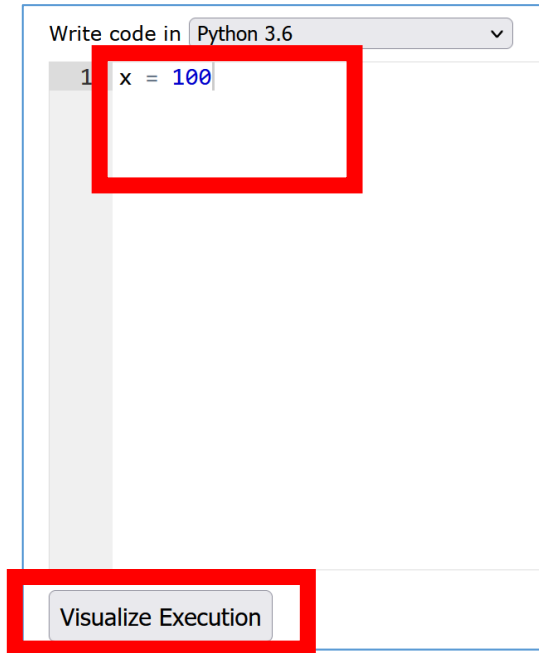
This tool helps you learn Python, JavaScript, C, C++, and Java programming by [visualizing code execution](#). You can use it to debug your homework assignments and as a supplement to online coding tutorials.

Start coding now in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Over 15 million people in more than 180 countries have used Python Tutor to visualize over 200 million pieces of code. It is the most widely-used program visualization tool for computing education.

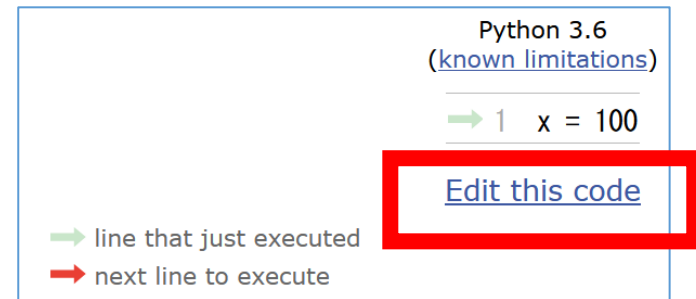
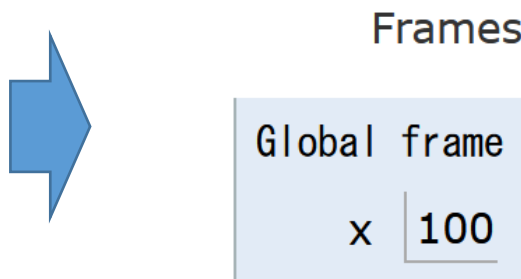
You can also embed these visualizations into any webpage. Here's an example showing recursion in Python:

Python Tutor でのプログラム実行手順



(1) 「**Visualize Execution**」をクリックして**実行画面**に切り替える

(2) 「**Last**」をクリック。



(3) 実行結果を確認する。

(4) 「**Edit this code**」をクリックして**編集画面**に戻る

Python Tutor 使用上の注意点①



- 実行画面で、次のような赤の表示が出ることもある →
無視してよい

過去の文法ミスに関する確認表示
邪魔なときは「Close」

Python Tutor: Visualize code in [Python](#), [JavaScript](#), [C](#), [C++](#), and [Java](#)

Python 3.6
([known limitations](#))

```
→ 1 x = 100
```

[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Step 1 of 1

[Customize visualization](#)

Frames Objects

You just fixed the following error:

```
1 x = 100!
```

SyntaxError: invalid syntax (<string>, line 1)

Please help us improve this tool with your feedback.
What misunderstanding do you think caused this error?

Submit Close [Hide all of these pop-ups](#)

Python Tutor 使用上の注意点②



「please wait ... executing」のとき，10秒ほど待つ。



→ 混雑しているときは，「Server Busy・・・」

というメッセージが出ることがある。

混雑している。少し（数秒から数十秒）待つと自

動で表示が変わる（変わらない場合には，操作を

もう一度行ってみる）

ステップ実行



- ステップ実行により、プログラム実行の流れをビジュアルに観察

Python 3.6

```
1 age = 30
2 if age <= 12:
3     print(500)
4 else:
5     print(1200)
```

[Edit this code](#)

Print output (drag lower right)

Frames

Global frame

age 30

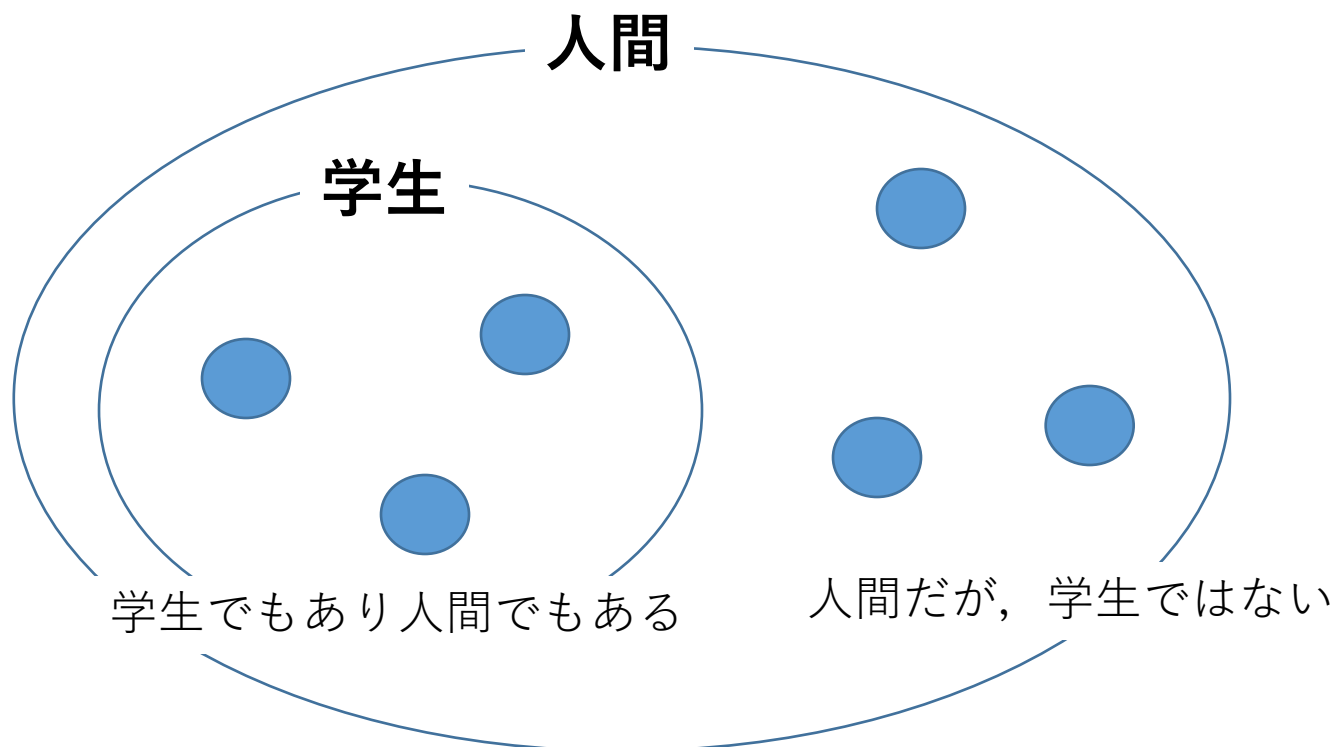
Step 3 of 3

8-1. クラスとオブジェクト

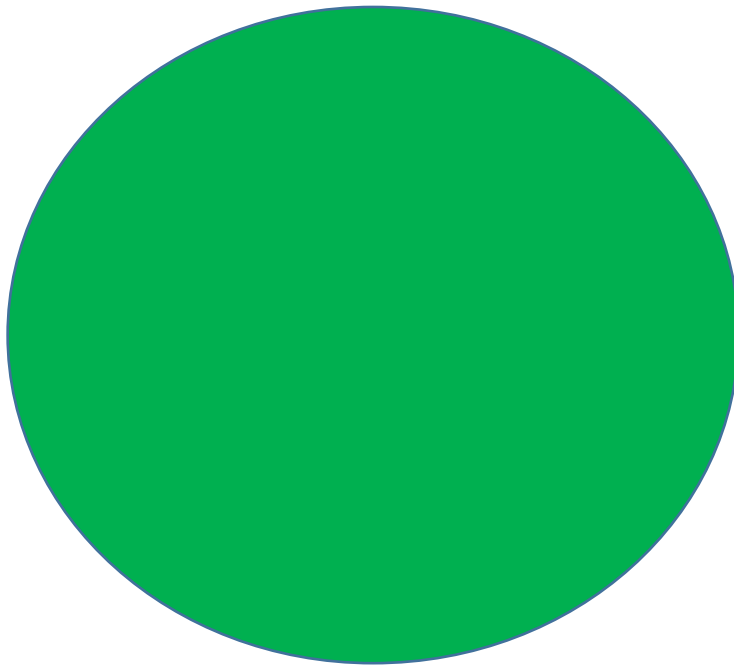
クラスとオブジェクト



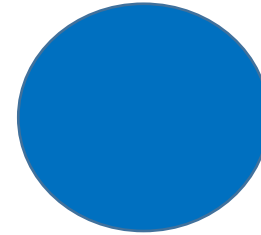
クラスは、同じ種類のオブジェクトの集まりと考えることができる



同じクラスの2つのオブジェクト



半径 3, 場所 (2, 4)
色 green



半径 1, 場所 (8, 10)
色 blue

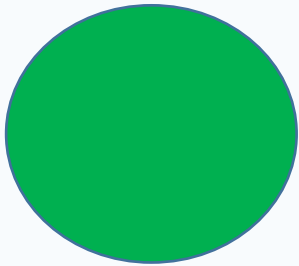
クラス Ball

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

- 2つのオブジェクトともに、**同じクラス Ball** と考えることができる
- **オブジェクト**は**属性**を持つ。
半径, 場所, 色などの属性

8-2. クラス定義, class, オブ ジェクト生成

クラス定義の例



```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
```

クラス名: Ball

メソッド: `__init__`, `printout`

属性: `x`, `y`, `r`, `color`

※ `__init__` は, オブジェクト生成のためのメソッド

クラス定義の例



```
1 class Ball: クラス名: Ball
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color          メソッド: __init__
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
                                         メソッド: printout
```

このクラス定義を使用した, **オブジェクト**の生成

a	8	10	1	"red"
b	2	4	3	"green"
	x	y	r	color

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
```

Python プログラム

メソッドアクセス



```
a.printout()  
b.printout()
```

Python プログラム

a や b **オブジェクト**
printout() **メソッド**
間を「.」で区切っている

演習

資料 : 22 ~ 27

【トピックス】

- クラス定義
- class
- オブジェクト生成
- メソッドアクセス

① Python Tutor のエディタで次のプログラムを入れる



クラス定義, オブジェクト生成, メソッドアクセス

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
```

クラス定義

```
10 a = Ball(8, 10, 1, "blue")
11 b = Ball(2, 4, 3, "green")
```

オブジェクト生成

```
12 a.printout()
```

```
13 b.printout()
```

メソッドアクセス

字下げも正確に



② 実行し，結果を確認する

メソッド printout による表示

Python 3.6
([known limitations](#))

```

1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9
10  a = Ball(8, 10, 1, "blue")
11  b = Ball(2, 4, 3, "green")
12  a.printout()
13  b.printout()

```

[Edit this code](#)

→ line that just executed

→ next line to execute

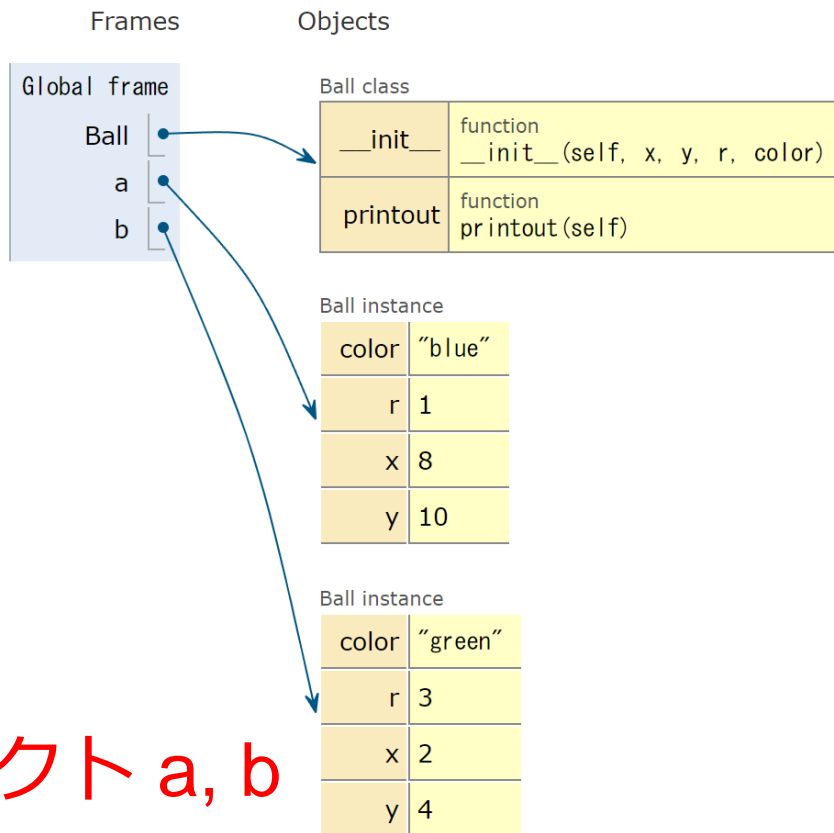
<< First < Prev Next > Last >>

Done running (23 steps)

[Customize visualization](#)

Print output (drag lower right corner to resize)

```
8 10 1 blue
2 4 3 green
```



オブジェクト a, b

「**Visual Execution**」をクリック。そして「**Last**」をクリック。結果を確認。
「**Edit this code**」をクリックすると，エディタの画面に戻る

③ 「First」 をクリックして, プログラム実行を先頭に戻す

Python 3.6
([known limitations](#))

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9
10 a = Ball(8, 10, 1, "blue")
11 b = Ball(2, 4, 3, "green")
12 a.printout()
13 b.printout()
```

[Edit this code](#)

→ line that just executed
→ next line to execute

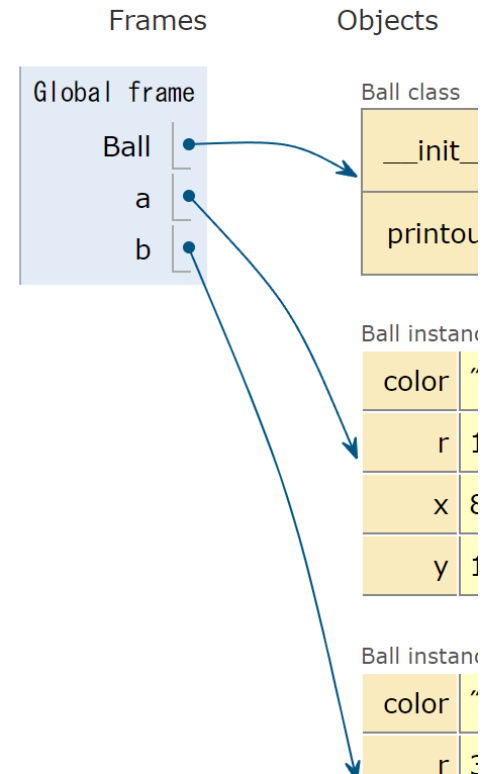
Progress bar and navigation controls:

- Progress bar: A horizontal bar with a slider on the right.
- Navigation buttons: << First, < Prev, Next >, Last >>

Done running (23 steps)

Print output (drag lower right corner to res)

```
8 10 1 blue
2 4 3 green
```



④ 「**Step 1 of 23**」 と表示されているので、
全部で、ステップ数は **23** あることが分かる
(ステップ数と、プログラムの行数は**違うもの**)

Python 3.6
([known limitations](#))

```
→ 1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9
10 a = Ball(8, 10, 1, "blue")
11 b = Ball(2, 4, 3, "green")
12 a.printout()
13 b.printout()
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First Previous Next Last >>

Step 1 of 23

Print output (drag lower right corn



Frames

Objects

⑤ 先頭に戻したので

- すべての**オブジェクト**は消えている
- **赤い矢印**は、**先頭に戻っている**

```
Python 3.6  
(known limitations)  
→ 1 class Ball:  
2     def __init__(self, x, y, r, color):  
3         self.x = x  
4         self.y = y  
5         self.r = r  
6         self.color = color  
7     def printout(self):  
8         print(self.x, self.y, self.r, self.color)  
9  
10 a = Ball(8, 10, 1, "blue")  
11 b = Ball(2, 4, 3, "green")  
12 a.printout()  
13 b.printout()
```

[Edit this code](#)

→ line that just executed

→ next line to execute



<< First < Prev Next > Last >>

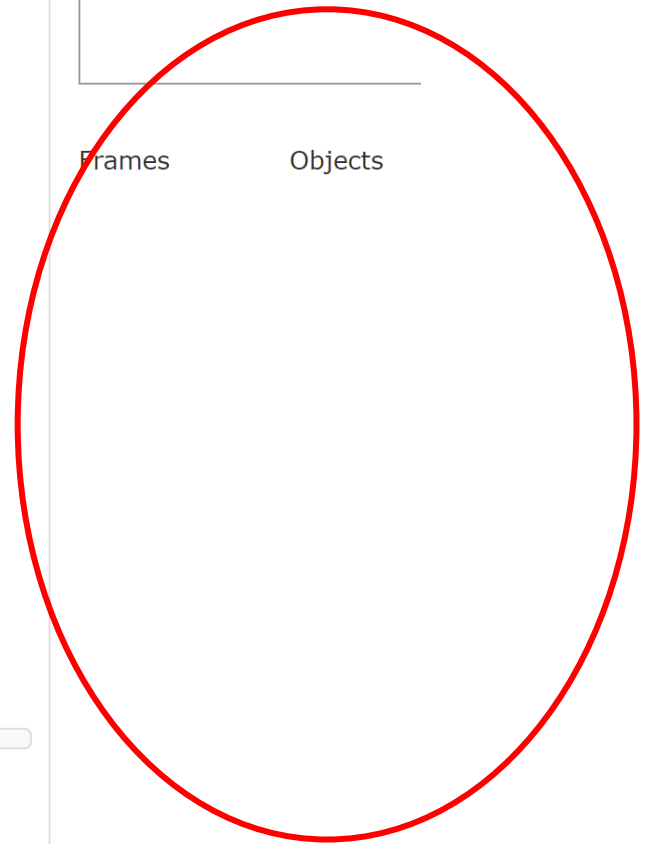
Step 1 of 23

Print output (drag lower right corn



Frames

Objects



⑥ **ステップ実行**したいので、「Next」をクリックしながら、**矢印の動きを確認**しなさい。



※ 「Next」 ボタンを何度か押し、それ以上進めなくなったら終了

あとで使うので、プログラムを消さずに残しておくこと

Python 3.6
([known limitations](#))

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9
10 → a = Ball(8, 10, 1, "blue")
11 → b = Ball(2, 4, 3, "green")
12 a.printout()
13 b.printout()
```

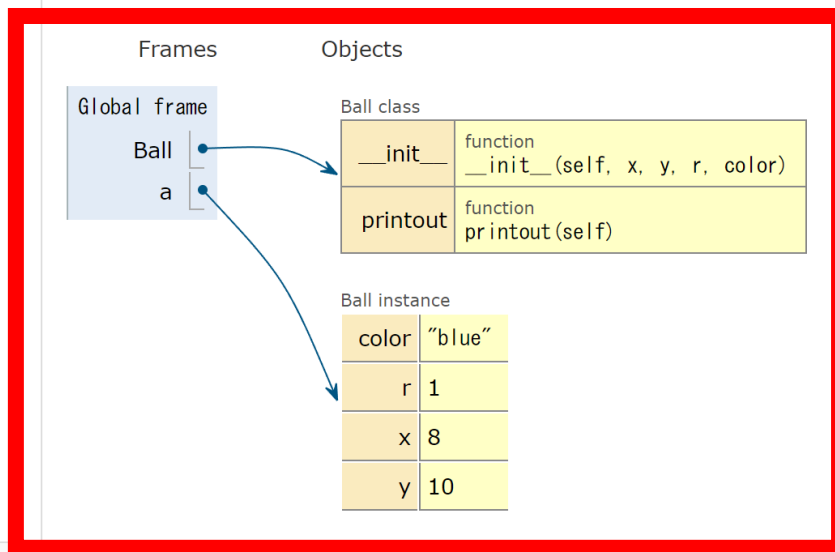
[Edit this code](#)

→ line that just executed
→ next line to execute

<< First < Prev Next > Last >>

Step 9 of 23

Print output (drag lower right corner to resize)



ジャンプしている

実行が進むと、
オブジェクトが更新される 27

まとめ



- **クラス定義**では、**クラス名の指定**と、**メソッド定義**を行う。

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
```

- キーワード

class **クラス**

def **定義**

__init__ オブジェクト生成のためのメソッド

8-3. メソッドアクセス, 属性 アクセス, self

メソッドと属性



- **メソッド**や**属性**は, **クラス**に属する
- **メソッド定義内**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

self による属性アクセス, メソッドアクセス



- **メソッド定義内**で, その**メソッドが所属するクラス**で定義された**属性**や**メソッド**にアクセスするときは **self + 「.」**

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
```

- メソッド外では「オブジェクト名」 + 「.」

```
12 a.printout()
13 b.printout()
```

演習

資料 : 33 ~ 35

【トピックス】

- self によるアクセス


```
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)
```

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.printout()
b.printout()
```

変更前

```
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)
    def dist(self):
        return self.x + self.y
```

メソッド dist
の定義

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.printout()
b.printout()
```

```
print(a.dist())
```

メソッドアクセス

変更後

① Python Tutor のエディタで次のプログラムを追加



```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9     def dist(self):
10        return self.x + self.y
```

```
12 a = Ball(8, 10, 1, "blue")
13 b = Ball(2, 4, 3, "green")
```

```
14 a.printout()
```

```
15 b.printout()
```

```
16 print(a.dist())
```

メソッド定義内の属性アクセス

メソッド外のメソッドアクセス

② 実行し，結果を確認する。

あとで使うので，プログラムを消さずに残しておくこと

Python 3.6
([known limitations](#))

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9     def dist(self):
10        return self.x + self.y
11
12 a = Ball(8, 10, 1, "blue")
13 b = Ball(2, 4, 3, "green")
14 a.printout()
15 b.printout()
16 print(a.dist())
```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev Next > Last >>

Done running (27 steps)

[tomize visualization](#)

Print output (drag lower right corner to resize)

```
8 10 1 blue
2 4 3 green
18
```

表示が増える

Frames

Global frame

- Ball
- a
- b

Objects

Ball class

__init__	function __init__(self, x, y, r, color)
dist	function dist(self)
printout	function printout(self)

Ball instance

color	"blue"
r	1
x	8
y	10

Ball instance

color	"green"
r	3
x	2
y	4

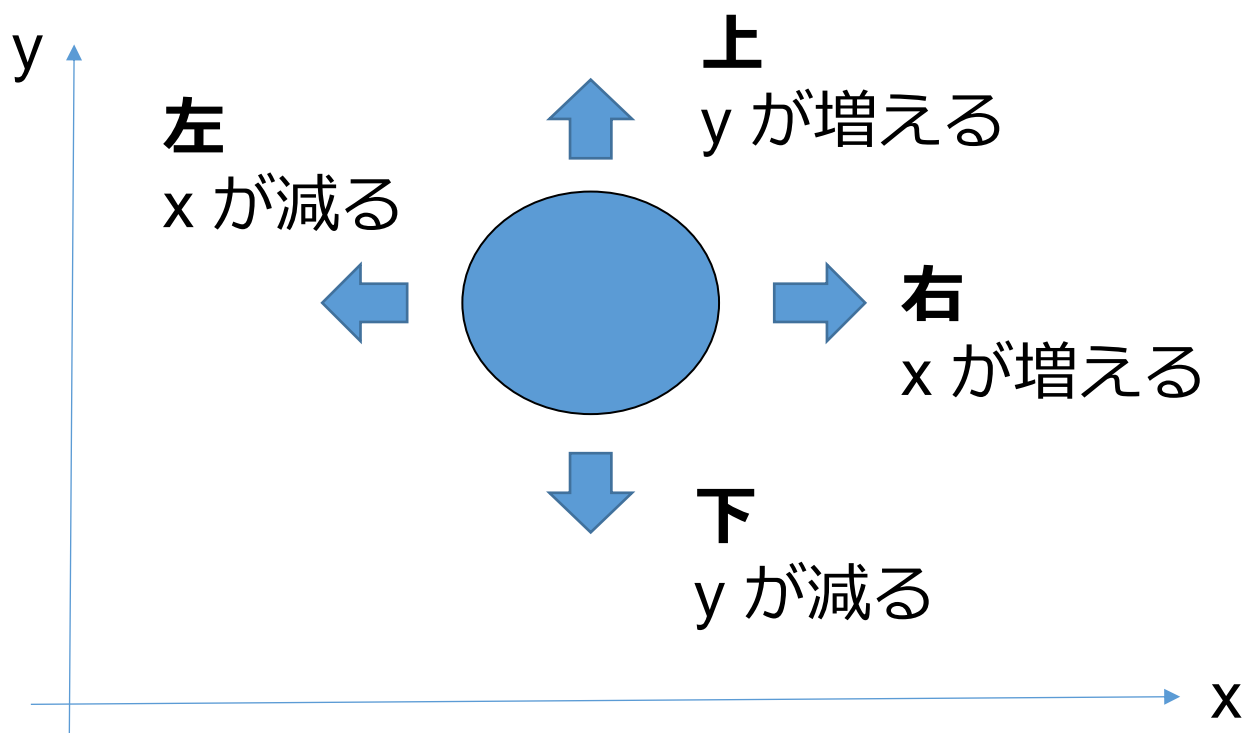
「**Visual Execution**」をクリック。そして「**Last**」をクリック。結果を確認。
「**Edit this code**」をクリックすると，エディタの画面に戻る

8-4. 演習

メソッド



- 上下左右の移動を考える
- **オブジェクト**の属性 x, y を増減
- そのためのメソッド `move` を定義



```
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)
    def dist(self):
        return self.x + self.y
```

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.printout()
b.printout()
print(a.dist())
```

変更前

```
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)
    def dist(self):
        return self.x + self.y
```

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

メソッド move
の定義

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
```

```
a.move(5, 5)
b.move(10, 10)
```

メソッドアクセス

```
a.printout()
b.printout()
print(a.dist())
```

変更後

演習

資料 : 40 ~ 45

【トピックス】

- メソッド定義
- メソッドアクセス

① Python Tutor のエディタで次のプログラムを入れる



オブジェクトを移動するメソッド move

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




② 実行し，結果を確認する

Python 3.6
([known limitations](#))

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

[Edit this code](#)

→ line that just executed

→ next line to execute

<< First < Prev Next > Last >>

Done running (37 steps)

[Customize visualization](#)

Print output (drag lower right corner to resize)

```
13 15 1 blue
12 14 3 green
28
```

表示が変わる

Frames

Objects

Global frame

Ball
a
b

Ball class

__init__	function __init__(self, x, y, r, color)
dist	function dist(self)
move	function move(self, xx, yy)
printout	function printout(self)

Ball instance

color	"blue"
r	1

x	13
y	15

値が変わる

Ball instance

color	"green"
r	3

x	12
y	14

値が変わる

「**Visual Execution**」をクリック。そして「**Last**」をクリック。結果を確認。
「**Edit this code**」をクリックすると，エディタの画面に戻る

演習問題

- 右に動かすためのメソッド `right` を定義
- 左に動かすためのメソッド `left` を定義
- `right` を使って, オブジェクト `a` を右に 5 動かす
- `Left` を使って, オブジェクト `b` を左に 10 動かす

演習問題の解答例



右に移動するメソッド right, 左に移動するメソッド left

```
1 class Ball:
2     def __init__(self, x, y, r, color):
3         self.x = x
4         self.y = y
5         self.r = r
6         self.color = color
7     def printout(self):
8         print(self.x, self.y, self.r, self.color)
9     def dist(self):
10        return self.x + self.y
11    def move(self, xx, yy):
12        self.x = self.x + xx;
13        self.y = self.y + yy;
14    def right(self, xx):
15        self.move(xx, 0)
16    def left(self, xx):
17        self.move(-xx, 0)
18
19 a = Ball(8, 10, 1, "blue")
20 b = Ball(2, 4, 3, "green")
21 a.move(5, 5)
22 b.move(10, 10)
23 a.right(5)
24 b.left(10)
25 a.printout()
26 b.printout()
27 print(a.dist())
```

実行し、結果を確認してみる

```

Python 3.6
(known limitations)
1  def printout(self):
8  print(self.x, self.y, self.r, self.color)
9  def dist(self):
10 return self.x + self.y
11 def move(self, xx, yy):
12 self.x = self.x + xx;
13 self.y = self.y + yy;
14 def right(self, xx):
15 self.move(xx, 0)
16 def left(self, xx):
17 self.move(-xx, 0)
18
19 a = Ball(8, 10, 1, "blue")
20 b = Ball(2, 4, 3, "green")
21 a.move(5, 5)
22 b.move(10, 10)
23 a.right(5)
24 b.left(10)
25 a.printout()
26 b.printout()
→ 27 print(a.dist())

```

[Edit this code](#)

line that just executed
next line to execute

<< First < Prev Next > Last >>

Done running (53 steps)

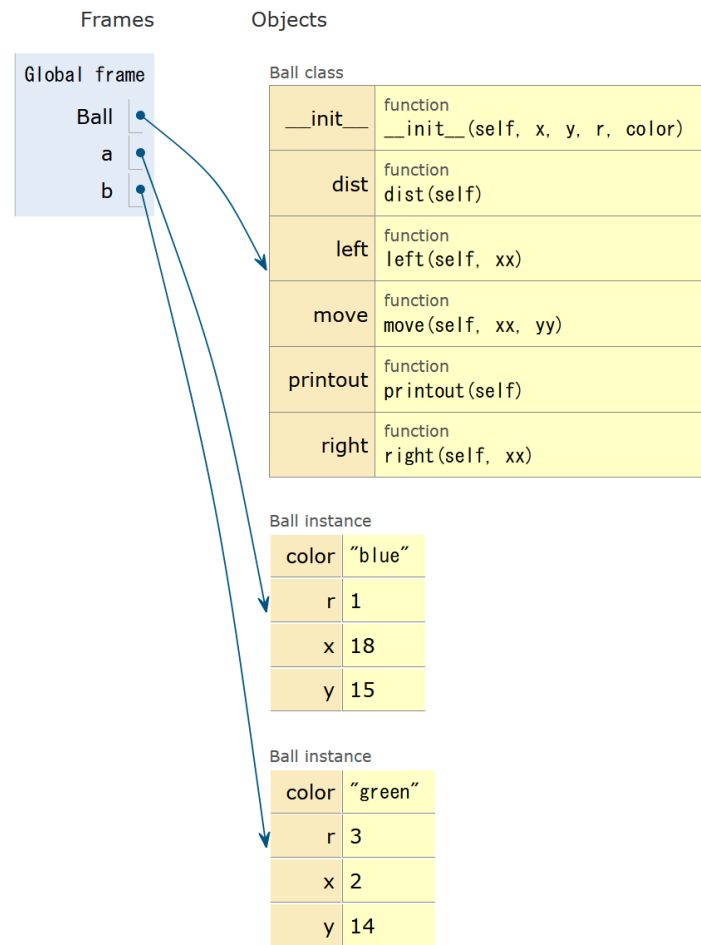
[tomize visualization](#)

Print output (drag lower right corner to resize)

```

18 15 1 blue
2 14 3 green
33

```



「**Visual Execution**」をクリック。そして「**Last**」をクリック。結果を確認。
「**Edit this code**」をクリックすると、エディタの画面に戻る

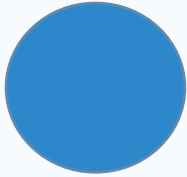
上や下に動かすためのメソッドを、
メソッド left, right を参考に作ってみなさい。

全体まとめ



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

```
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)
```

クラス定義のプログラム

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
```

オブジェクト生成のプログラム

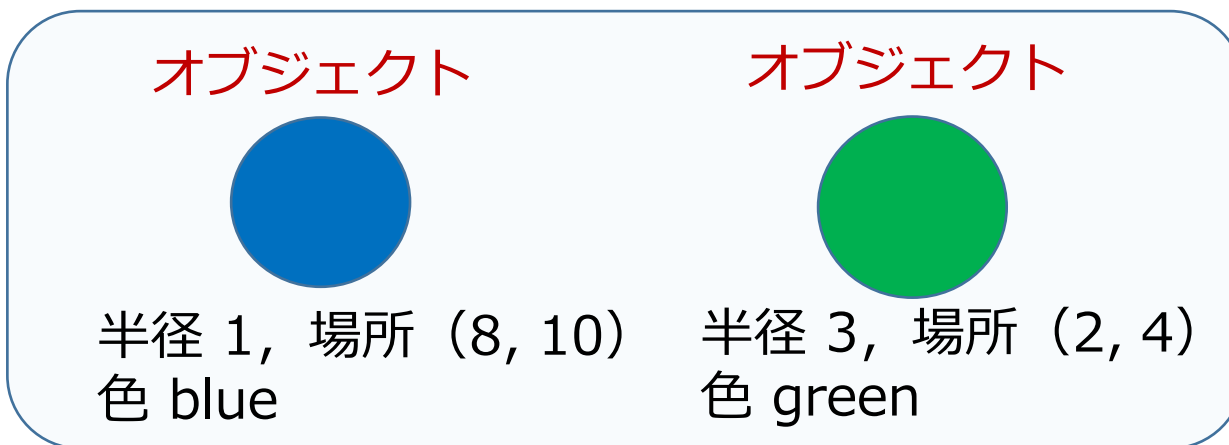
```
a.printout()
b.printout()
```

メソッドアクセスのプログラム

全体まとめ



- **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 Ball

- **メソッド定義内で、そのメソッドが所属するクラスで定義された**属性**や**メソッド**にアクセスするときは **self + 「.」****

```
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)
```

Python 関連ページ



- Python まとめページ

<https://www.kkaneko.jp/tools/man/python.html>

- Python 入門（スライド資料とプログラム例）

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

- Python プログラミングの基本（スライド資料とプログラム例）

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

- Python プログラム例

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

- 人工知能の実行（Google Colaboratory を使用）

<https://www.kkaneko.jp/ai/ni/index.html>

- 人工知能の実行（Python を使用）（Windows 上）

<https://www.kkaneko.jp/ai/deepim/index.html>


```
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)
```

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.printout()
b.printout()
```

```
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)
    def dist(self):
        return self.x + self.y
```

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.printout()
b.printout()
print(a.dist())
```

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

a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.move(5, 5)
b.move(10, 10)
a.printout()
b.printout()
print(a.dist())
```

```
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)
    def dist(self):
        return self.x + self.y
    def move(self, xx, yy):
        self.x = self.x + xx;
        self.y = self.y + yy;
    def right(self, xx):
        self.move(xx, 0)
    def left(self, xx):
        self.move(-xx, 0)
```

```
a = Ball(8, 10, 1, "blue")
b = Ball(2, 4, 3, "green")
a.move(5, 5)
b.move(10, 10)
a.right(5)
b.left(10)
a.printout()
b.printout()
print(a.dist())
```