



# po-4. 関数, 抽象化

(Python プログラミング体験学習, 全9回)

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

金子邦彦





## アウトライン

4-1 復習

4-2 式の抽象化と関数

4-3 関数内の変数

4-4 関数呼び出し

- **Python Tutor** に慣れる.
- **抽象化**, **関数** について理解を深める.



# 4-1. 復習

# Python のプログラム例



- 変数  $x$  の値を 100 に変化させる

**「 $x = 100$ 」**

Write code in Python 3.6

```
1 x = 100
```

## • 代入

「 $x = 100$ 」のように書くと、 $x$  の値が 100 に変化する

- **式**の実行結果として、値が得られる
- 式の中に、変数名を書くことができる

```
import math  
r = 3  
print(r * r * math.pi)
```

式

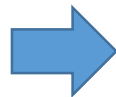
# 式の実行結果として、値が得られる



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

複数の**式**

Print output (drag lower right)



```
110.000000000000001
165.0
440.000000000000006
```

実行結果

## 実習の指示

- 資料： **8** ~ **12**
- 次のことを理解しマスターする
  - 式の結果として値が得られる

# 実習



① **ウェブブラウザ**を起動する

② **Python Tutor** を使いたいのので、次の URL を開く

**<http://www.pythontutor.com/>**

※ Internet Explorer でうまく動かない場合がある

→ うまく動かないときは Google Chrome を試してください

※ 途中で 「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）

※ 日本語モードはない。英語で使う





### ③ 「Python Tutor」をクリック

← → ↻ ⓘ 保護されていない通信 | pythontutor.com ☆ APP 2

## VISUALIZE CODE AND GET LIVE HELP

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

[Python Tutor](#) (created by [Philip Guo](#)) helps people overcome a fundamental barrier to learning programming: understanding what happens as the computer runs each line of code.

Write code in your web browser, see it visualized step by step, and get live help from volunteers.

Related services: [Java Tutor](#), [C Tutor](#), [C++ Tutor](#), [JavaScript Tutor](#), [Ruby Tutor](#)

**Over five million people in more than 180 countries** have used Python Tutor to visualize over 100 million pieces of code, often as a supplement to textbooks, lectures, and online tutorials.

[Visualize your code and get live help now](#)



Get live help!

Start private chat

These Python Tutor users are asking for help right now. Please volunteer to help!  
user\_041 from Singapore, Singapore needs help with Python3 - 2 people chatting - requested 12 minutes ago)  
user\_91c from Vélizy-Villacoublay, France needs help with Python3 - [click to help](#) minutes ago)  
user\_985 from Seattle, Washington, US needs help with Python3 - [click to help](#) (-

「Python 3.6」になっている

Write code in Python 3.6

```
1 |
```

エディタ

Help improve this tool by completing a [short user survey](#)

実行のためのボタン

Visualize Execution

Live Programming Mode

hide exited frames [default] inline primitives but don't nest objects [default]

draw pointers as arrows [default]

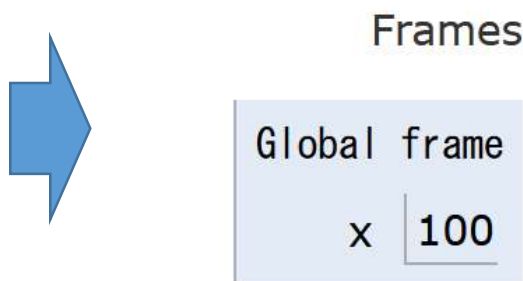
# Python Tutor でのプログラム実行手順



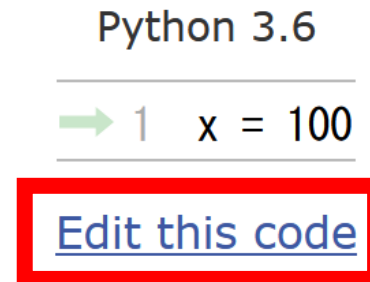
(1) 「**Visualize Execution**」をクリック。



(2) 「**Last**」をクリック。



(3) 結果を確認する。



(4) 「**Edit this code**」をクリックして戻る



④ 次のプログラムを実行し，結果を確認しなさい

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

Print output (drag lower right)

```
110.00000000000000001
165.0
440.00000000000000006
```



## 4-2. 式の抽象化と関数



## 問いかけ

- プログラミングでの根本問題は  
何でしょうか？
- プログラミングの一番の基礎は  
何でしょうか？

# 式の抽象化



$$100 * 1.1$$

$$150 * 1.1$$

$$400 * 1.1$$



$$a * 1.1$$

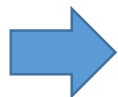
**変数** a を使って, 複数の**式**を1つにまとめる  
(**抽象化**)

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
```



```
def foo(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



## 私の意見

- プログラミングでの根本問題は  
何でしょうか？  
**誤り（バグ）の無いプログラムの  
作成**
- プログラミングの一番の基礎は  
何でしょうか？
  - **抽象化を行うこと**.
  - **抽象化により、繰り返し同じこ  
とを書くことが減り**, バグを防  
げる.
  - プログラムの変更（消費税率  
10%変更）も簡単に.

## 実習の指示

- 資料：21
- 次のことを理解しマスターする
  - 式の抽象化と関数



次のプログラムを実行し，結果を確認しなさい

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

Print output (drag lower right)

```
110.00000000000000001  
165.0  
440.0000000000000006
```

表示を確認



## 4-3. 関数内の変数



## 問いかけ

- 関数では、変数が使用されることがあります。
- 「関数内で使用される変数は、関数の実行終了とともに、自動で消える」は正しいですか？



- 関数 foo では、変数 a が使用されています
- 「関数内で使用される変数 a は、関数の実行終了とともに、自動で消える」は正しいですか？

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
7 print(foo(p))
```





## 実習の指示

- 資料：26～27
- 次のことを理解しマスターする  
「関数内で使用される変数は、関数の実行終了とともに、自動で消える」

# 実習



次のプログラムを実行し，結果を確認しなさい

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
7 print(foo(p))
```

Print output (drag lower right corner to res

```
132.0  
220.00000000000003
```

Frames      Objects

Global frame	function foo(a)
foo	
p 200	

実行結果を確認



### Python 3.6

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
→ 7 print(foo(p))
```

[Edit this code](#)

cutted  
ute

<< First   < Prev   Next >   Last >>

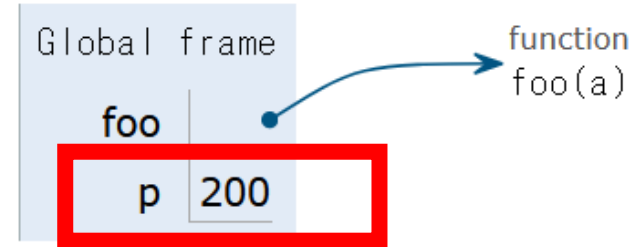
Done running (11 steps)

Print output (drag lower right corner to r

```
132.0  
220.0000000000000003
```

Frames

Objects

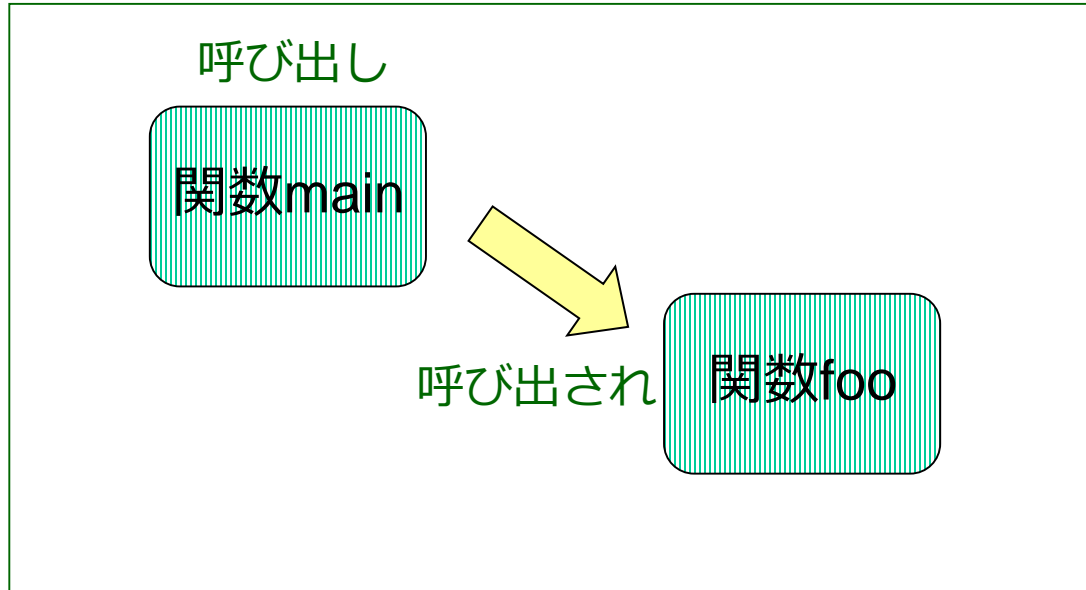


- 変数 p は残っている
- 変数 a は自動的に消えている



## 4-4. 関数呼び出し

# 関数呼び出し





## 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



## 実習の指示

- 資料：32～39
- 次のことを理解しマスターする
  - Python Tutor でのステップ実行の操作手順
  - 関数呼び出しにおけるジャンプ
  - 関数内で使用される変数が消えるタイミング



# ① 「First」 をクリックして，最初に戻しなさい

Python 3.6

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
→ 7 print(foo(p))
```

[Edit this code](#)

ited  
e

<< First

< Prev

Next >

Last >>

Done running (11 steps)

Print output (drag lower right corner to re

```
132.0  
220.0000000000000003
```

Frames

Objects

Global frame

foo

p

200

function  
foo(a)





② 「**Step 1 of 11**」 と表示されているので、  
全部で、ステップ数は **11** 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

```
→ 1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
7 print(foo(p))
```

[Edit this code](#)

uted  
:e

<< First

< Prev

Next >

Last >>

Step 1 of 11

Print output (drag lower right cor

Frames

Objects



### ③ プログラム実行を最初に戻したので

- すべての**変数**は消えている
- **赤い矢印**は、**先頭行に戻っている**

Python 3.6

```
→ 1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
7 print(foo(p))
```

[Edit this code](#)

Print output (drag lower right corner)

Frames      Objects

Step 1 of 11

<< First   < Prev   Next >   Last >>



④ **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



見どころ

foo との間で

**ジャンプ**するところ

Python 3.6

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
→ 5 print(foo(p))  
→ 6 p = 200  
7 print(foo(p))
```

[Edit this code](#)

<< First < Prev **Next >** Last >>

Step 7 of 11

Print output (drag lower right corner to re

132.0

Frames

Objects

Global frame

foo

p

120

function

foo(a)

⑤ 終わったら、もう一度、  
て、最初に戻しなさい

「**First**」をクリックし



Python 3.6

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
→ 7 print(foo(p))
```

[Edit this code](#)

uted  
ite

<< First

< Prev

Next >

Last >>

Done running (11 steps)

Print output (drag lower right corner to re

```
132.0  
220.000000000000003
```

Frames

Objects

Global frame

foo

p

200

function  
foo(a)



⑥ もう一度、ステップ実行しなさい。

「Next」をクリックしながら、

- ・ 赤の矢印の動きを確認しなさい。
- ・ 変数 `a` が現れるのは、どういうタイミングですか？

The screenshot shows the Python 3.6 interactive shell with the following code:

```
Python 3.6
1 def foo(a):
2     return a * 1.1
3
4 p = 120
5 print(foo(p))
6 p = 200
7 print(foo(p))
```

The code is being executed step-by-step. The current step is line 5, `print(foo(p))`, which is highlighted with a green arrow. The variable inspector shows the following state:

Frames	Objects
Global frame	function foo(a)
foo	120
p	120

The variable `a` in the `foo` frame is circled in red, indicating its presence during the execution of the function `foo`.

関数 `foo` の実行中は、  
変数 `a` が現れる



# ⑦ 終わったら「**Edit self code**」をクリックして、元の画面に戻る

Python 3.6

```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
→ 7 print(foo(p))
```

[Edit this code](#)

cuted  
jte

<< First < Prev Next > Last >>

Done running (11 steps)

Print output (drag lower right corner to re

```
132.0  
220.0000000000000003
```

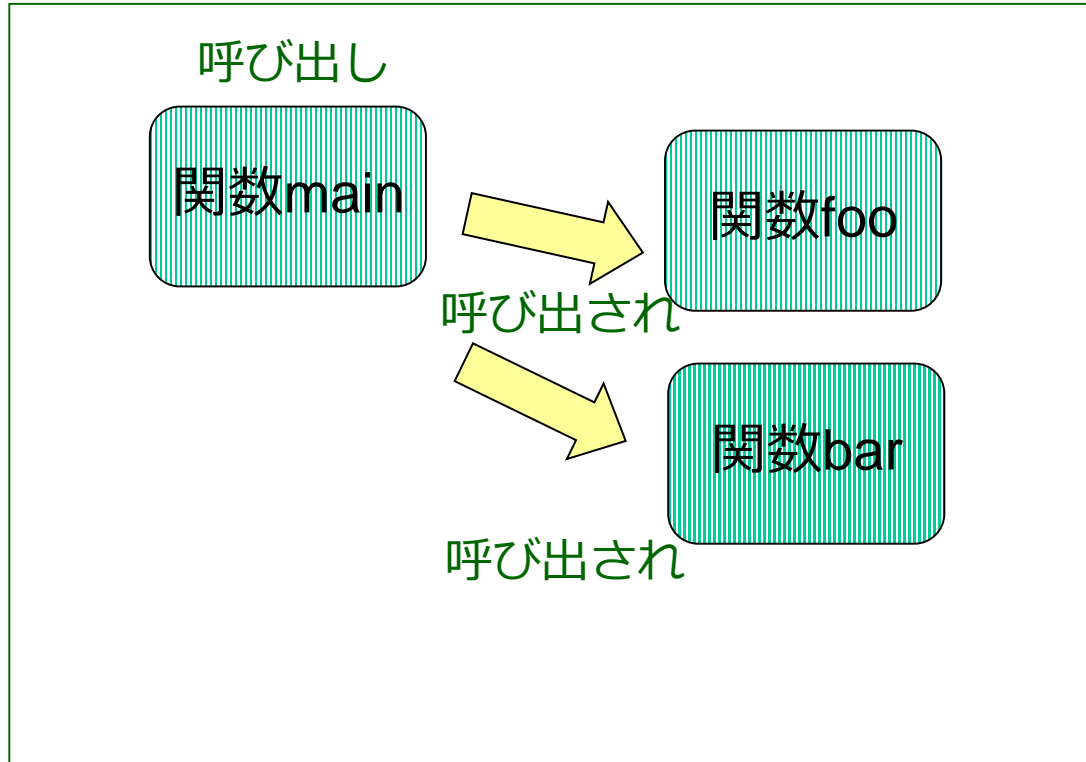
Frames	Objects
Global frame	
foo	function foo(a)
p	200

⑧ 次のように書き換えて，②から⑦と同じことを繰り返さない



```
1 def foo(a):  
2     return a * 1.1  
3  
4 p = 120  
5 print(foo(p))  
6 p = 200  
7 print(foo(p))  
8 p = 400  
9 print(foo(p))
```

# 関数呼び出し





# 式の抽象化



$$5 * 100 * 1.1$$

$$12 * 100 * 1.1$$



類似した複数の**式**

$$a * 100 * 1.1$$

1つの**式**に**抽象化**

$$a * 100$$

$$a * 1.1$$

2つの**式**に**抽象化**  
できるという考え方も

# 単価 150円で, 税率 10% のときのプログラム例

## 変数 `c` は個数のつもり



```
1 def foo(a):
2     return a * 1.1
3
4 def bar(a):
5     return a * 100
6
7 c = 5
8 print(foo(bar(c)))
9 c = 12
10 print(foo(bar(c)))
```

5個は 550円  
12個は 1320円

Print output (drag)

```
550.0
1320.0
```

# 全体まとめ



- **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
- **関数の中でのみ**使用される**変数**は、**関数の終了**とともに**自動で消える**
- **関数の評価値**は **return** で返す