プログラミング実習I クラス5 (井村担当)

知能・機械工学課程 井村 誠孝

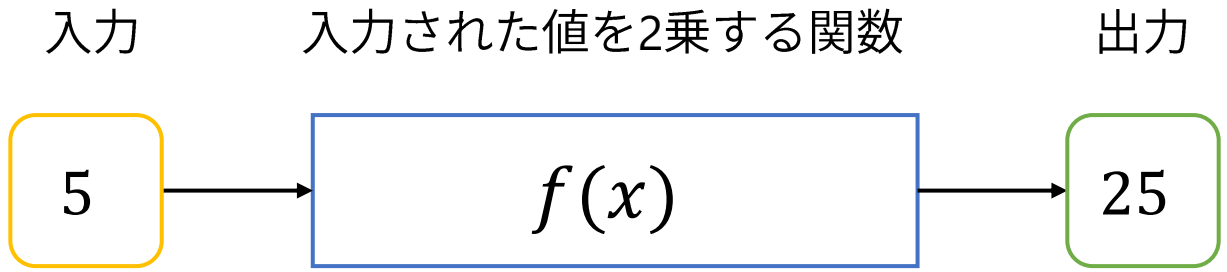
m.imura@kwansei.ac.jp

関数(function)とは

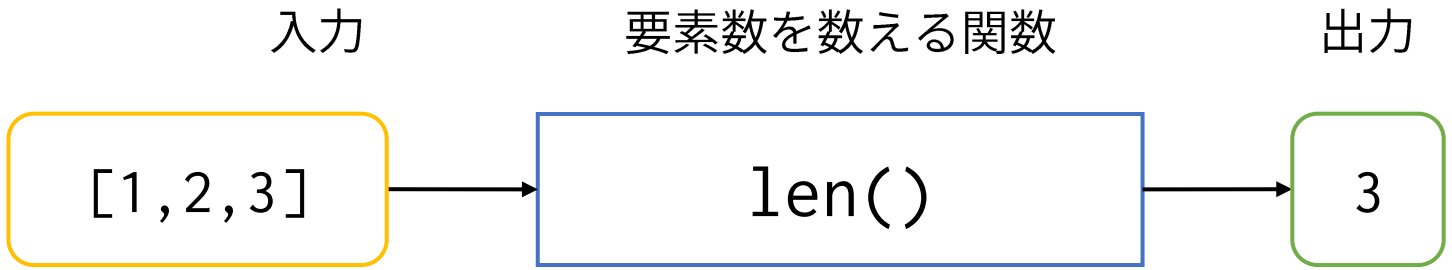
- 数学的には入力に対して出力が決まるもの
- コンピュータ言語では入出力が定まったひとまとまりの処理
  - 入力や出力はあるときもないときもある．
- 何かの機能(function)を有している魔法のブラックボックス
  - 例: 関数 len()
    - 関数中で行われている処理の詳細を使う側は知らないが，とにかく，要素数を調べてほしいリスト型データを与えると，個数を数えて教えてくれる．

数学の関数とPythonの関数

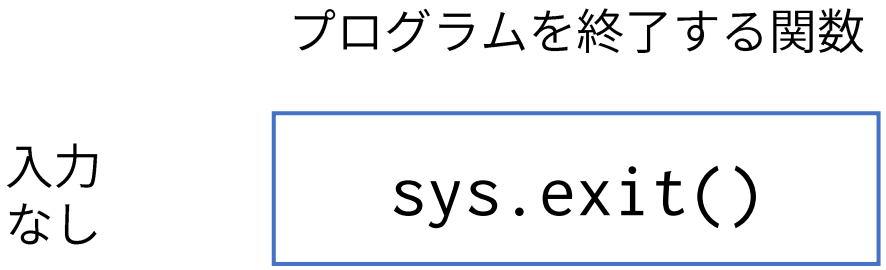
数学



Python



確かに似てる



えーと…

関数 len() がもしなかったら…

- for文を使ってカウンタを1ずつ増加

```
test_list = [1, 2, 3]
length = 0
for i in test_list:
    length += 1
```

- 関数を使うと

```
test_list = [1, 2, 3]
length = len(test_list)
```

利点

- for文を自身で書かなくてよい．
- 変数 length の値の意味が明瞭
- 同じ処理(この場合，リストの要素数を数える)を他でも使える．
  - 修正が必要なときにあちこちで同じ修正をしなくて済む．
- 自分で実装するより高速な可能性がある．

関数はどういうときに便利か

- ある処理を複数回行う場合
  - 同じソースコードを複数回書くのは面倒，かつ，変更するときが大変．間違える可能性も高くなる．
- ある処理が複数の処理から成り立っている場合
  - 長い手順からなる処理を，段階ごとに分割し関数として実装すると，見通しがよくなる．
  - 処理の入力と出力が明確になり，その処理のみに必要な変数などを関数内に隠蔽することができる．

関数の使い方は皆さん既に知っている

- これまでに使った関数の例
  - 組み込み関数

`len()` `print()` `int()` `input()` `range()`

  - モジュール

`math.sin()` `sys.exit()`

後でモジュールを自作する

- 関数の使い方は既に知っている．
  - ソースコード中に関数名と入力=引数(ひきすう)を書く．
  - 出力=戻り値 or 返り値は変数に代入したり，直接使用したりする．
  - 使いたい関数によっては，モジュールをimportする必要がある．
    - 例: 数学関数であれば import math

余談: 関数から返ってくる値を何と呼ぶか/書くか

- 「戻り値」vs「返り値」vs「戻値」vs「返値」

関数を作成する

関数を自作する方法

関数定義は
defで開始

```
def 関数名(引数...):
    ブロック
```

関数を定義する

```
def func():
    i = 3
```

データを返す

```
def func():
    i = 3
    return i
```

return 文で
戻り値を返す

引数を持つ

引数が変数として渡される

```
def func(v):
    i = v + 3
    return i
```

関数を呼び出す

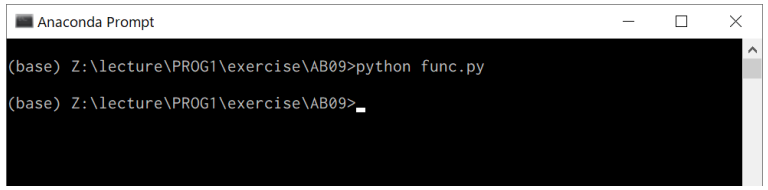
```
func()
```

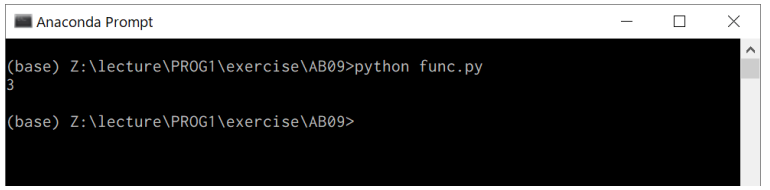
関数を呼び出す

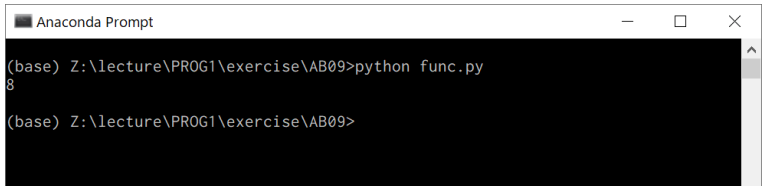
```
ret = func()
print(ret)
```

関数を呼び出す

```
ret = func(5)
print(ret)
```







もうすこし意味のある機能のある関数の例

引数なし
戻り値あり

```
import random
def shoot():
    rand1 = random.randint(1, 6)
    return rand1

print(shoot())
```

引数あり(2つ)
戻り値あり

```
def plus(x, y):
    z = x + y
    return z

print(plus(1, 2))
print(plus(3, 4))
```

関数から関数を呼び出して使う例

```
import random

def shoot():
    rand1 = random.randint(1, 6)
    return rand1

def plus(x, y):
    z = x + y
    return z

def two_dices():
    r = plus(shoot(), shoot())
    return r

print(two_dices())
```

複雑な引数や戻り値も可能

引数あり(2つのタプル)
戻り値あり

```
import math

def [          ](p, q):
    dx = p[0] - q[0]
    dy = p[1] - q[1]
    d = math.sqrt(dx * dx + dy * dy)
    return d

print([          ]((1, 2), (3, 4)))
```

Q. これはどのような働きをする関数でしょうか.

引数あり(リスト)
戻り値あり(リスト)

```
def [          ](x_list):
    y_list = []
    for x in x_list:
        y_list.append(x * x)
    return y_list

print([          ]([1, 2, -3]))
```

Q. これはどのような働きをする関数でしょうか.

自作の関数の命名方法

- Pythonでの関数名の原則: 英小文字(+数字)を _ (アンダーバー)で接続した名前にする.
- 例
  - `print()`
  - `calc_score()`
  - `very_very_long_function_name()`
- PEP 8 https://pep8-ja.readthedocs.io/ja/latest/ の「命名規約」を確認してください.

言語が異なると流儀が変わる場合もあるので注意.
長いものには巻かれておく.

関数とモジュール

関数をモジュールにする

- モジュール: 関数や変数をまとめたもの.
- スクリプトファイル .py がひとかたまりのモジュールになる.
  - モジュール名 = ファイル名から拡張子 .py を除いたもの
- 実行する際は，モジュール名.関数名() で指定する.

```
import random
def shoot():
    rand1 = random.randint(1, 6)
    return rand1
```

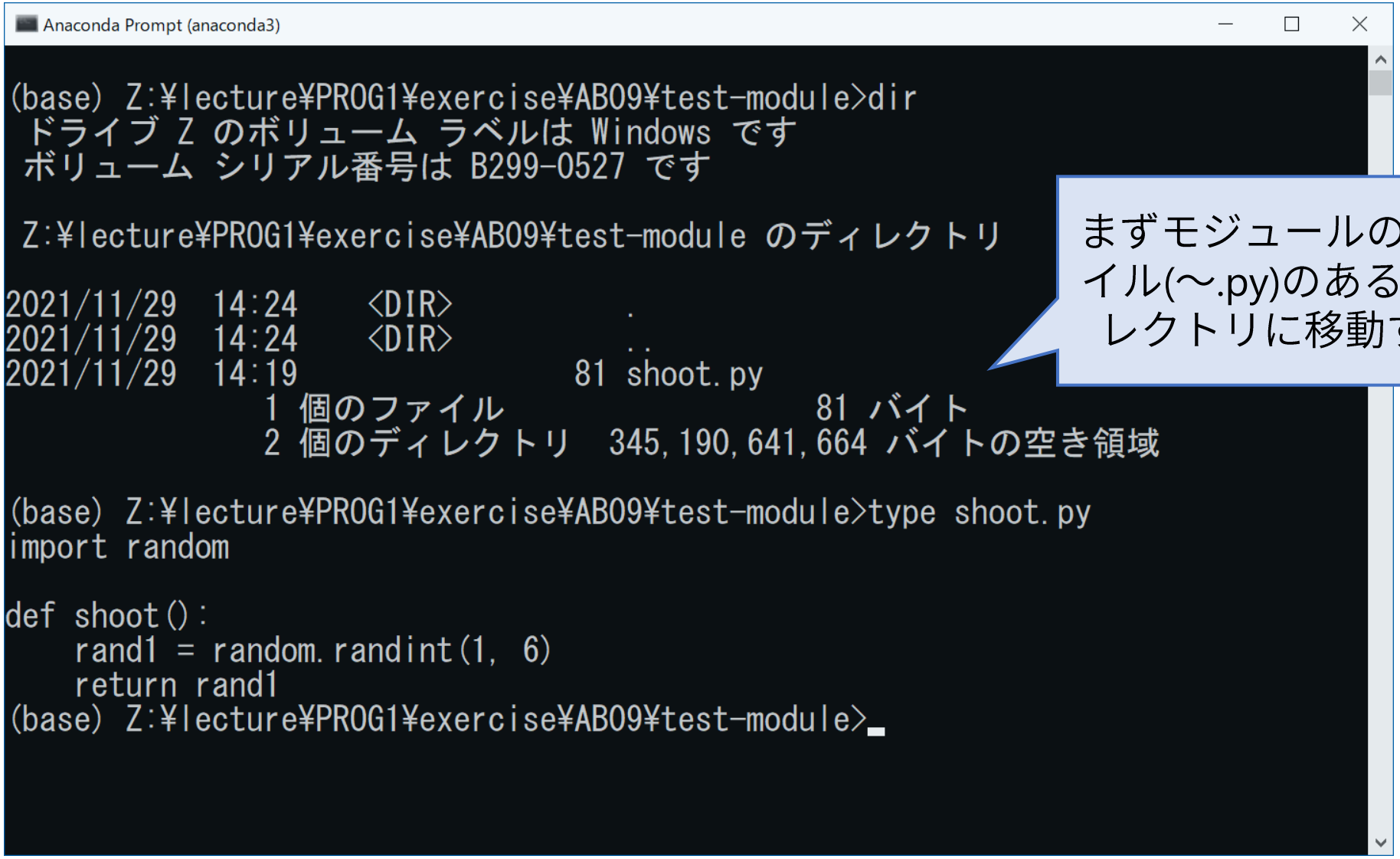
保存 → shoot.py → 利用

```
import shoot
print(shoot.shoot())
```

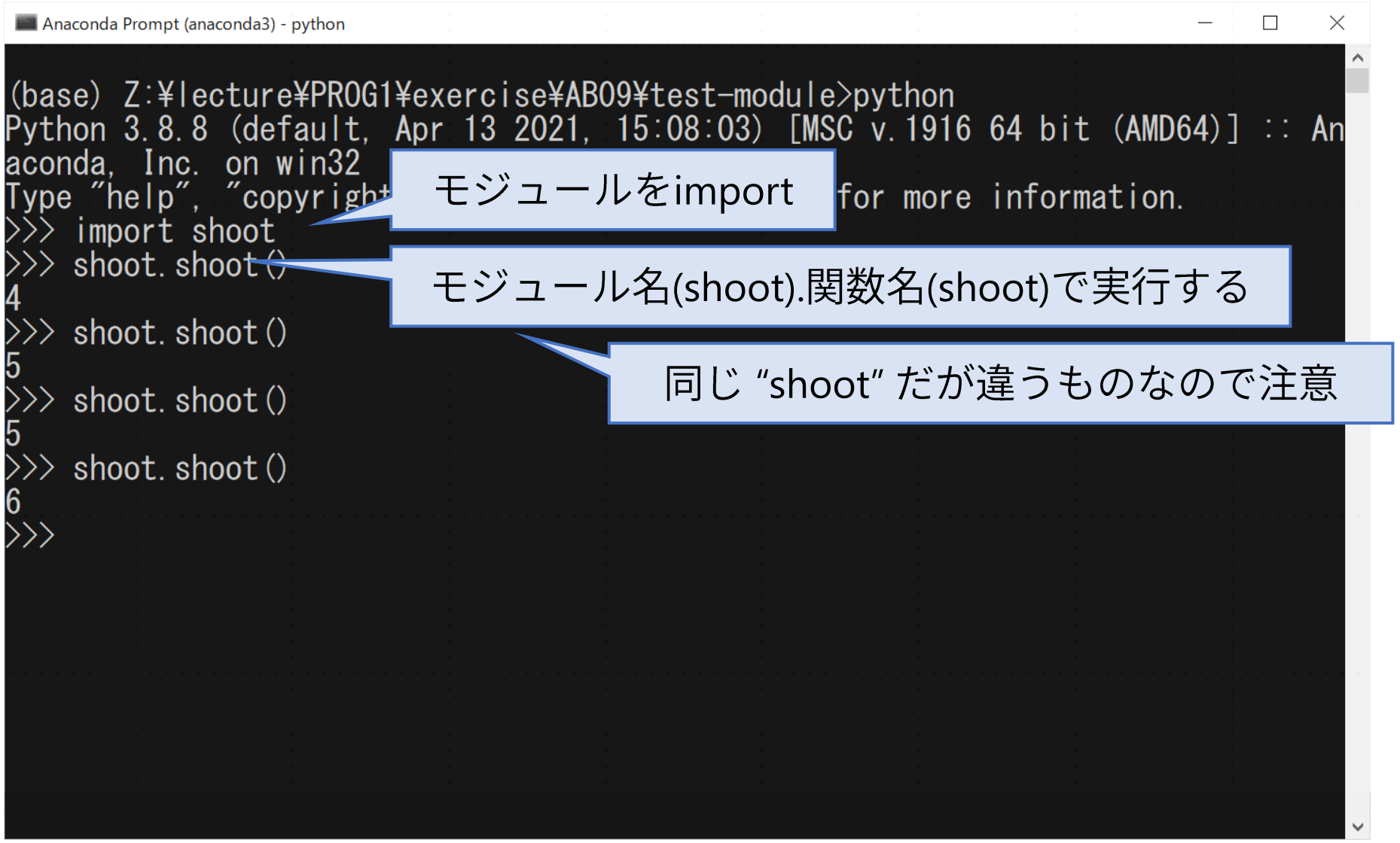
別のスクリプト

モジュール名と関数名が同じケースは，
Python標準ライブラリでもよく見られる.

インタラクティブシェルからの利用



インタラクティブシェルからの利用



モジュール名を省略する (使い方に注意)

- from モジュール名 import 関数名 とすると，モジュール名を省略して関数を呼び出すことができる．

通常のimport

```
import math

print(math.log(2))
```

fromを用いたimport

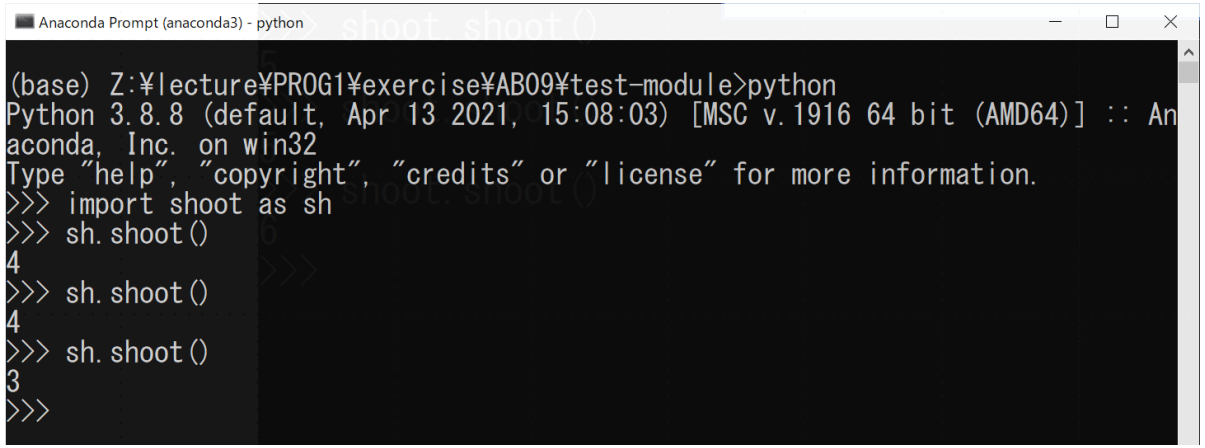
```
from math import log

print(log(2))
```

- 注意: from モジュール名 import * は原則として使用しない．
  - モジュール内で定義されている全ての関数(だけでなく定数なども)がモジュール名なしで使える．
    →プログラマが把握していない関数や変数の定義がなされる．
    →予期しない挙動をする可能性がある．

モジュールを違う名前で取り扱う

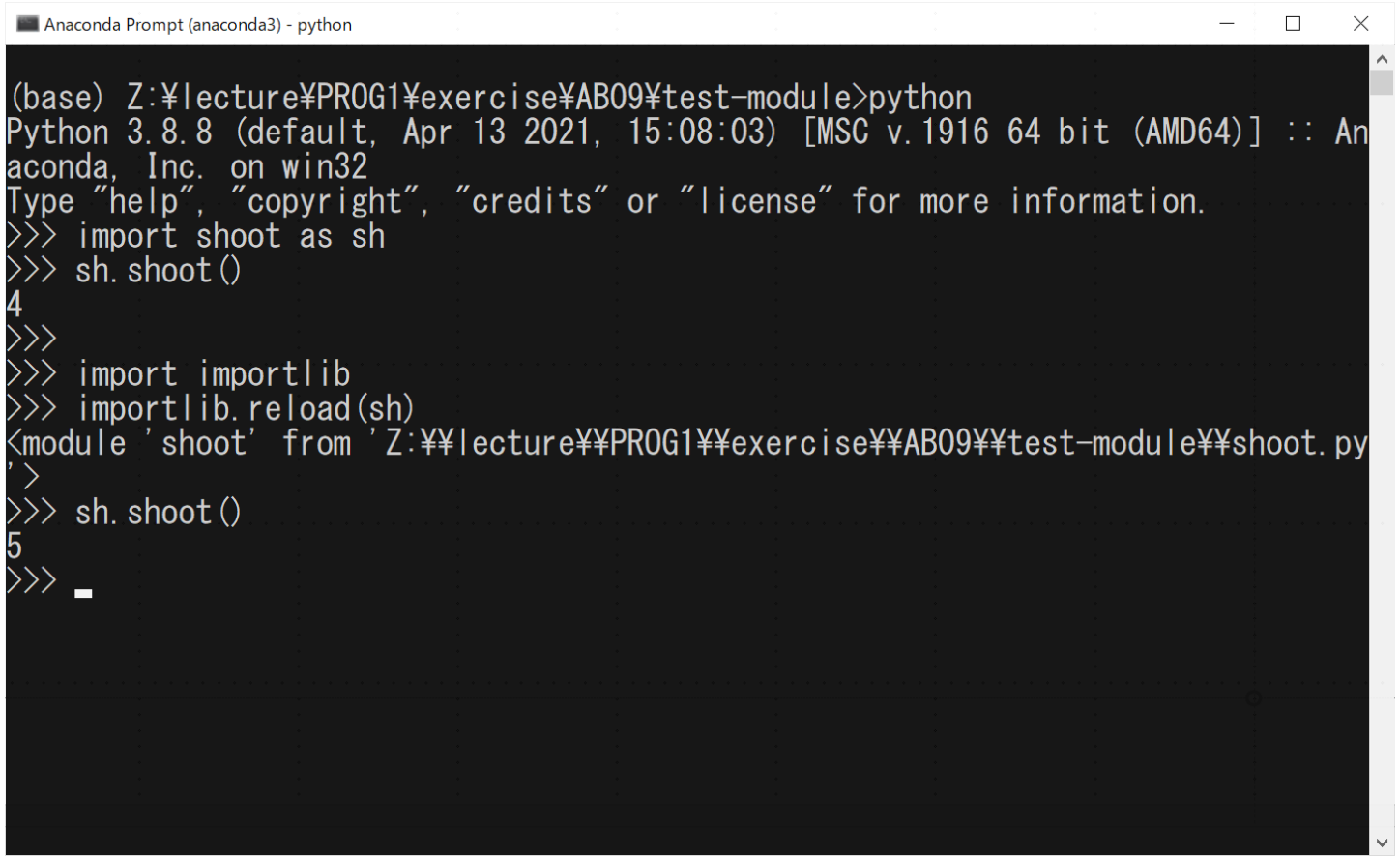
- import モジュール名 as 名前 とすると，モジュール名(長くて何回もソースコードに現れると鬱陶しい場合がある)の代わりに名前を使うことができる．
- よく見られる例
  - import numpy as np
  - import matplotlib.pyplot as plt
- 当然，自作のモジュールでも利用可能
  - import shoot as sh

NumPyは数値演算モジュール
matplotlibはグラフ描画モジュール
どちらもとても便利



インタラクティブシェルでのモジュールの再import

- インタラクティブシェルで利用しているモジュールについて，元のスクリプトファイル(モジュール名.py)を書き換えた場合は，importlib.reload()をする必要がある．



ローカル変数

- 関数定義内で代入される変数は，関数内でのみ利用可能なローカル変数である．
  - 代入せず参照するだけであれば関数外の変数に対しても行えるが，原則として使用しないようにする．

関数内と関数外のデータのやりとりは
引数と戻り値によって行う

悪い例と良い例

悪い例

```
import turtle

def center_circle():
    kame.penup()
    kame.forward(200)
    kame.left(90)
    kame.pendown()
    kame.circle(200)
    kame.left(90)
    kame.penup()
    kame.forward(200)
    kame.pendown()

kame = turtle.Turtle()
kame.shape('turtle')
kame.shapesize(2, 2, 3)

center_circle()
turtle.done()
```

関数外の変数kameを
参照している
→汎用性がない

良い例

```
import turtle

def center_circle(target):
    target.penup()
    target.forward(200)
    target.left(90)
    target.pendown()
    target.circle(200)
    target.left(90)
    target.penup()
    target.forward(200)
    target.pendown()

kame = turtle.Turtle()
kame.shape('turtle')
kame.shapesize(2, 2, 3)

center_circle(kame)
turtle.done()
```

引数で渡された
変数を参照している
→汎用性がある

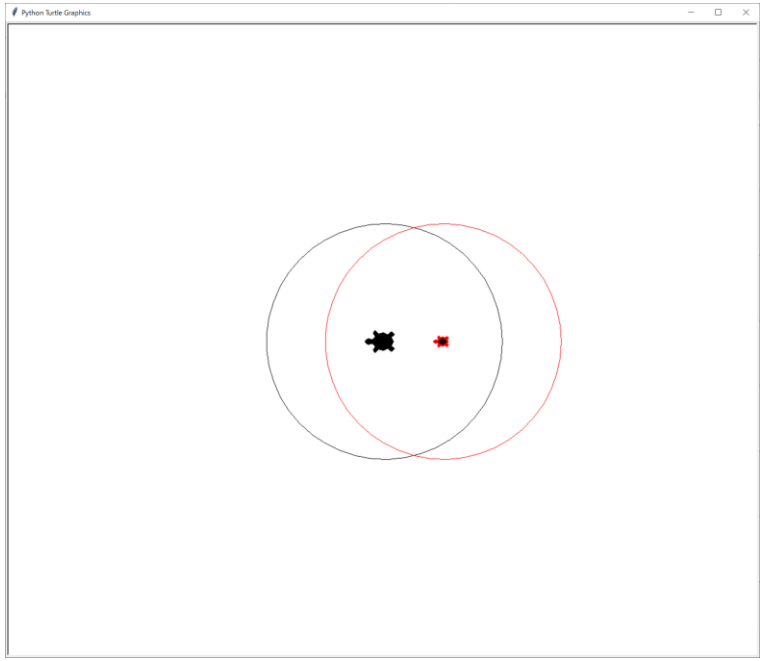
出入口は
引数と戻り値

引数を使ってやりとりすれば，亀が2匹いても大丈夫

```
import turtle

def center_circle(target):
    target.penup()
    target.forward(200)
    target.left(90)
    target.pendown()
    target.circle(200)
    target.left(90)
    target.penup()
    target.forward(200)
    target.pendown()
```

kameki　kametaro

```
kamekichi = turtle.Turtle()
kamekichi.shape('turtle')
kamekichi.shapesize(2, 2, 3)
center_circle(kamekichi)

kametaro = turtle.Turtle()
kametaro.shape('turtle')
kametaro.shapesize(1, 1, 3)
kametaro.pencolor('red')
kametaro.penup()
kametaro.forward(100)
kametaro.pendown()
center_circle(kametaro)

turtle.done()
```



半径を引数で与える

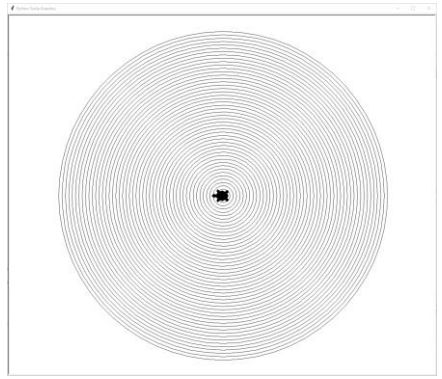
```
import turtle

def center_circle(target, r):
    target.penup()
    target.forward(r)
    target.left(90)
    target.pendown()
    target.circle(r)
    target.left(90)
    target.penup()
    target.forward(r)
    target.pendown()

kame = turtle.Turtle()
kame.shape('turtle')
kame.shapesize(2, 2, 3)

center_circle(kame, 100)
turtle.done()
```

謎の定数200が3箇所に書かれていた(本当はよろしくない)が解消されている点もいいね!

半径100の円

```
import turtle

def center_circle(target, r):
    target.penup()
    target.forward(r)
    target.left(90)
    target.pendown()
    target.circle(r)
    target.left(90)
    target.penup()
    target.forward(r)
    target.pendown()

kame = turtle.Turtle()
kame.shape('turtle')
kame.shapesize(2, 2, 3)

for r in range(10, 500, 10):
    center_circle(kame, r)
turtle.done()
```

大量に描いてみた



順番に渡される

変数rの値は順番に変わる

引数のデフォルト値

```
import turtle

def center_circle(target, r=150):
    target.penup()
    target.forward(r)
    target.left(90)
    target.pendown()
    target.circle(r)
    target.left(90)
    target.penup()
    target.forward(r)
    target.pendown()

kame = turtle.Turtle()
kame.shape('turtle')
kame.shapesize(2, 2, 3)

center_circle(kame)
turtle.done()
```

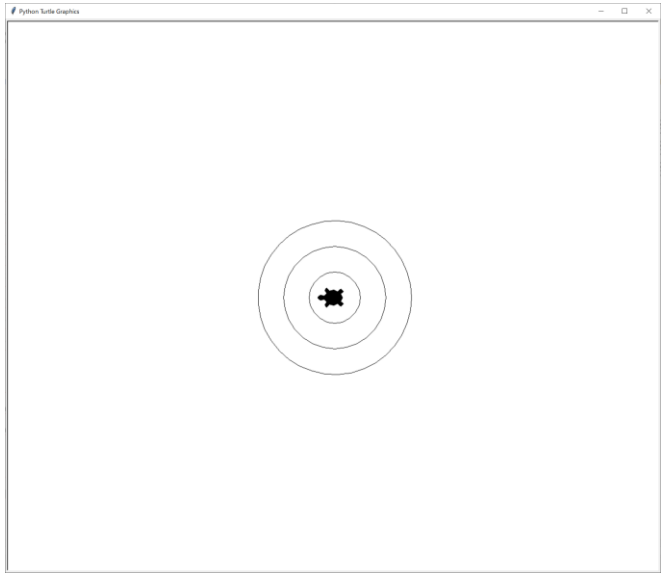
仮引数に続いて
=でデフォルト値を指定する

この例:
省略された場合r=150を使う

第2引数が
省略されている

例

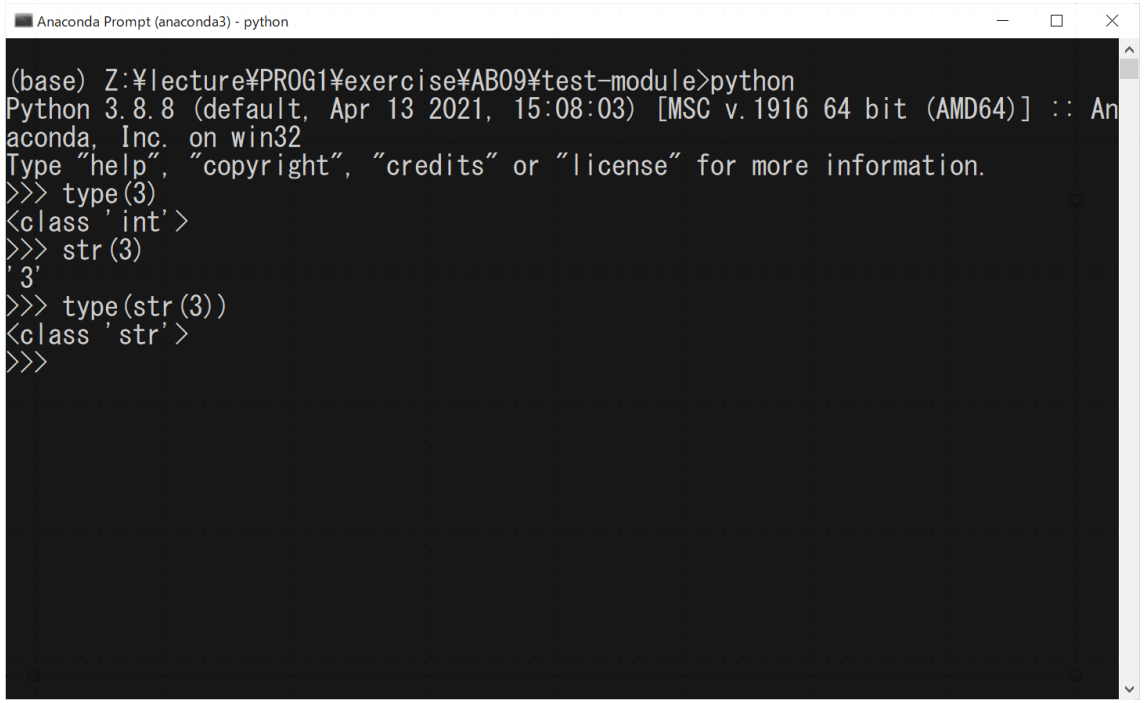
```
center_circle(kame)          省略した(r=150)
center_circle(kame, 100)     省略しない
center_circle(kame, 50)      省略しない
```

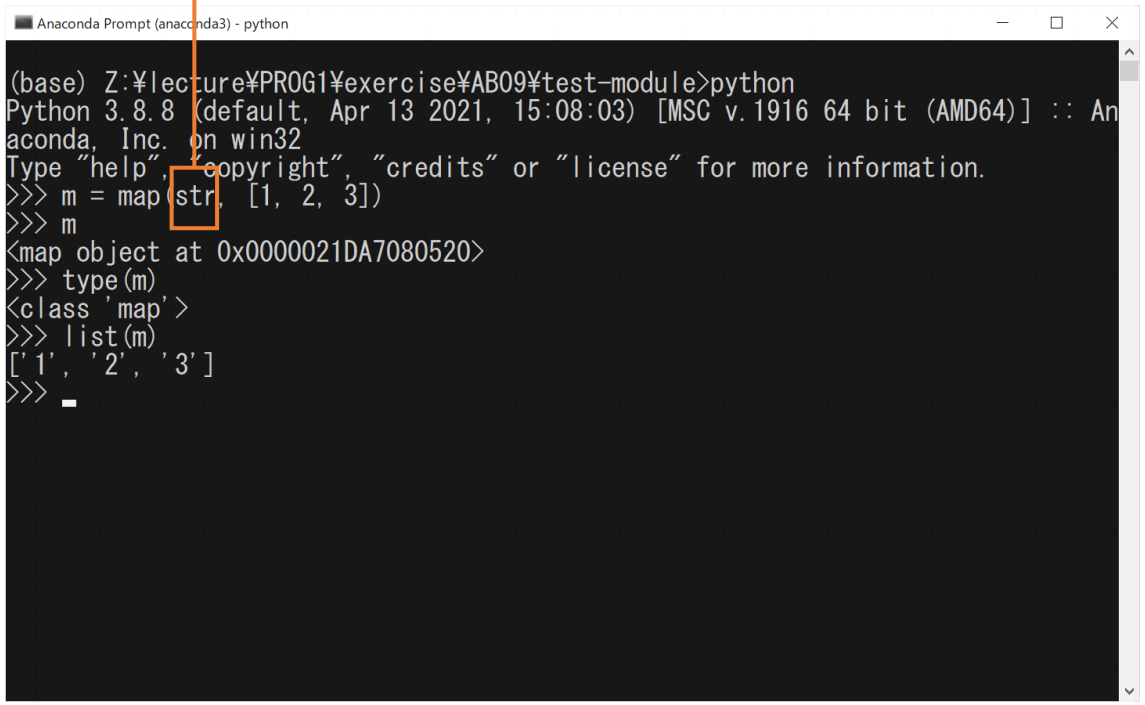


関数を引数に取る関数

- 組み込み関数 map(func, iter)
  - iter の要素のそれぞれに関数 func() を適用した結果を返す．
    - 正確には要求がなされると適用して返す．
- 教科書の例

関数の名前が()なしで登場している点に注目





再帰呼び出し

- 関数が自分の中で自分自身を呼び出す

```
def sum_from1_to(n):
    if n > 1:
        return sum_from1_to(n - 1) + n
    else:
        return 1

print(sum_from1_to(3))
```

まず sum_from1_to(3) を呼び出す．
n=3なので，ifの条件はTrue
値を返そうとするが，sum_from1_to(2)を
呼び出す必要がある．
というわけでsum_from1_to(2)を呼び出す．
n=2なので，またもifの条件はTrue
値を返そうとするが，sum_from1_to(1)を
呼び出す必要がある．
というわけでsum_from1_to(1)を呼び出す．
n=1なので，ついにifの条件はFalse
そこで1を返す．
すると1+2=3を返す．
すると3+3=6を返す．
6が戻り値として得られたので，表示する．