

The Kyoto College of
Graduate Studies
for Informatics

kcg.edu

コンピュータプログラミング概論

第9回 eラーニング資料

安 平勲

h_an@kcg.ac.jp

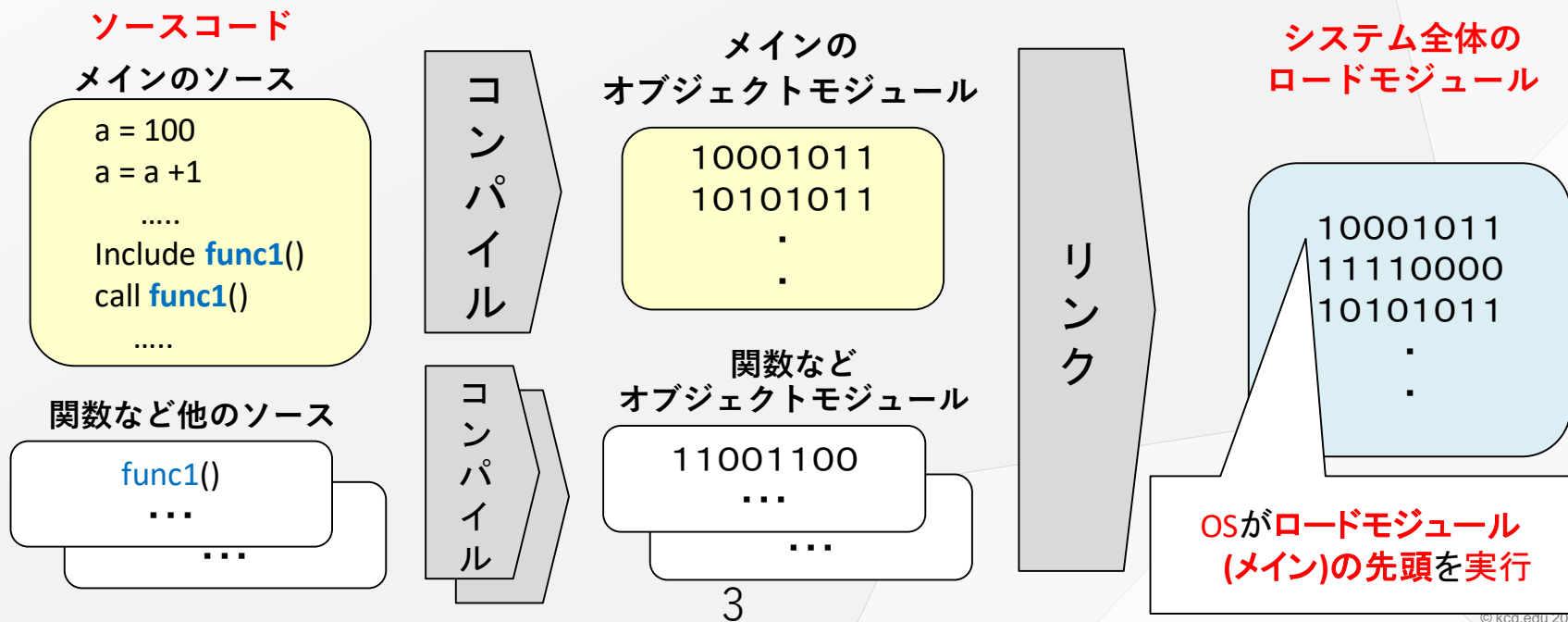
モジュールとパッケージ

パッケージ

- 規模の大きな**システム**は、複数のプログラム・モジュールに機能分割されて提供される。
一般的に、 1モジュール：100行～300行が目安
- **パッケージ**は複数のPythonモジュールからなる一つのシステムを提供する仕組み
- **パッケージ**はPython のモジュール名前空間を“ドット付きモジュール名”を使って構造化する手段
 - ⇒ 要は、変数名や関数名が他のプログラム・モジュールと重複しても大丈夫な仕組み

コンパイル言語のシステムが動く仕組み

- コンパイル言語では、メインプログラムと関数プログラムを**それぞれコンパイル**し、全てのオブジェクトモジュールをリンクすることで、OS（WindowやMacなど）動作する**一つのロードモジュール**が作られる



Pythonのシステムが動く仕組み

- インタプリタ言語であるPythonでは、**インタプリタ**が、呼ばれたソースを1行ずつ翻訳・実行する。他ファイルのモジュール化された関数が呼ばれれば、インタプリタはモジュールを特定し、そのソースを1行ずつ翻訳・実行する

ソースコード

```
a = 100  
a = a + 1  
....  
from folderX import mylib  
mylib.func1()  
...
```

mylib.py

```
def func1()  
...  
def funcN()
```

folderX

翻訳 : a=100

翻訳 : a=a+1

...

モジュールmylibの場所を特定

mylibの中のfunc1を呼び出し

翻訳 : func1を実行

...

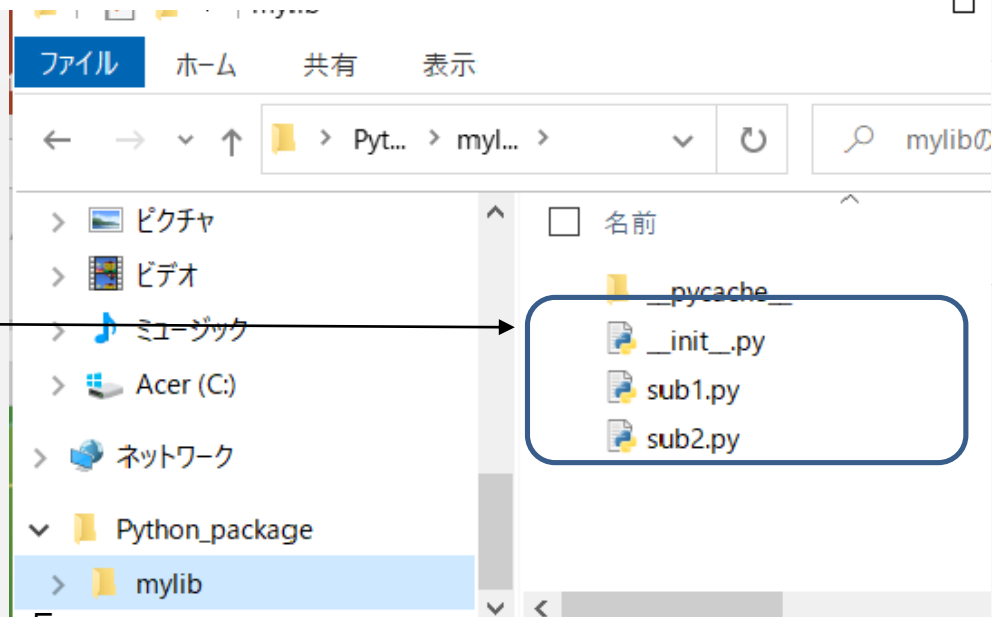
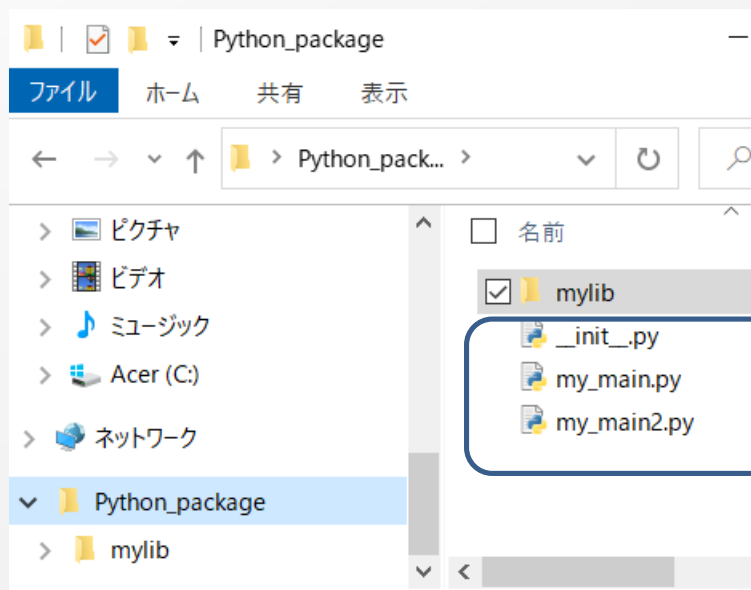
(Python等)
インタプリタ

ソースコードを1行ずつ
翻訳・実行

パッケージの例 (1 / 4)

■ モジュールをディレクトリ構造で配置すればパッケージ

- ルートディレクトリ下にファイルフォルダ配置
- フォルダ内に**モジュール**を配置する。空の `__init__.py` も付加



パッケージの例 (2 / 4)

- my_main.pyがパッケージmylibにあるモジュールsub1.pyとsub2.pyに定義された関数を呼ぶ前に **from パッケージ名 import モジュール名**
- 関数を呼び出す時、関数名の前に **モジュール名.** を付ける

my_main.py

```
# mylibパッケージのsub1, sub2モジュールインポート
from mylib import sub1, sub2

print('sub1モジュールのfunc_a関数呼出し')
sub1.func_a()

print('sub1モジュールのfunc_b関数呼出し')
sub1.func_b()

# sub2モジュールの同じ名前の関数func_aを呼出す
print('sub2モジュールのfunc_a関数呼出し')
sub2.func_a()
```

sub1.py

```
def func_a():
    print('sub1モジュールのfunc_aが呼ばれた')

def func_b():
    print('sub1モジュールのfunc_bが呼ばれた')
```

sub2.py

```
def func_a():
    print('sub2モジュールのfunc_aが呼ばれた')
```

パッケージの例（3 / 4）

- sub1とsub2ともに同名のfunc_a関数があるが、**モジュール名が異なる**ので区別して呼べる

my_main.pyの実行結果

```
Reloaded modules: mylib, mylib.sub2, mylib.sub1
sub1モジュールのfunc_a関数呼出し
sub1モジュールのfunc_aが呼ばれた
sub1モジュールのfunc_b関数呼出し
sub1モジュールのfunc_bが呼ばれた
sub2モジュールのfunc_a関数呼出し
sub2モジュールのfunc_aが呼ばれた
```


パッケージの例 (4 / 4)

- **from** で **パッケージ名.モジュール名** を指定して、**import** では **関数名** を指定すれば、関数呼び出し時には **モジュール名** は不要

my_main2.py とその実行結果

```
from mylib.sub1 import func_a  
  
print('sub1モジュールのfunc_a関数呼出し')  
func_a()
```

Reloaded modules: mylib, mylib.sub1,
sub1モジュールのfunc_a関数呼出し
sub1モジュールのfunc_aが呼ばれた

- ✓ from パッケージ名 import モジュール名
 - 呼び出し時：モジュール名.関数名 ()
- ✓ from パッケージ名.モジュール名 import 関数名
 - 呼び出し：関数名 ()

ビット演算

2 進数

- 整数は、通常は10進数で表すが、コンピュータの内部では2進数に変換されて保持されている
- 2進数は1と0（電流のonとoff）で数表現する。この単位を**ビット(bit)**と呼ぶ。また、8ビットを1**バイト(Byte)**と呼ぶ

```
# 2進数
print(0b1)
print(0b10)
print(0b100)
print(0b1000)
print(0b10000)
# 8進数
print(0o1)
print(0o10)
# 16進数
print(0x1)
print(0x10)
```

1
2
4
8
16
1
8
1
16

Pythonでの表記方法

- **0bn** : 2進数 n:0~1
- **0on** : 8進数 n:0~7
- **0xn** : 16進数 n:0~F

算術演算と論理演算(ビット演算)

■ 1 と 1 の和 ?

- (1) **10**進数の**算術演算**では $\cdot \cdot 1 + 1 = 2$ $10 + 10 = 20$
- (2) **2**進数の**算術演算**では $\cdot \cdot 1 + 1 = 10$ $10 + 10 = 100$
- (3) **論理**(ビット)**演算**では $\cdot \cdot 1 + 1 = 1$ $10 + 10 = 10$



コンピュータ (IC回路) はビット演算から算術演算を実現

ビット演算子

a=0b0101, b=0b0001の場合

bit演算子	意味	例	結果
&	論理積 (AND)	a & b	0b0001
	論理和 (OR)	a b	0b0101
^	排他的論理和 (XOR)	a ^ b	0b0100
~	ビット反転	~a	(次ページ)
<<	左シフト	a<<1	0b1010
>>	右シフト	a>>1	0b0010

ビット演算

```
a = 0b0101
b = 0b0001
print(a&b)
print(a|b)
print(a^b)
print(~a)
print(a<<1)
print(a>>1)
```

1
5
4
-6
10
2

✓ '&' '|' '^' は集合 (set) の演算子と同意

- ✓ ~ (チルダ) は符号反転。1の補数になる
- ✓ <<は2の乗算×2になる
- ✓ >>は2の除算÷2になる (余りは切捨て)