po-7. モジュール，標準ライブラリ，算法（アルゴリズム）

トピックス：モジュール，インポート，import，サブモジュール，パッケージ，標準ライブラリ，算法（アルゴリズム）

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

① モジュール

プログラムの共通部分を別ファイルにできる機能.

② 算法（アルゴリズム）

※　コンピュータは，ある定まった手順により問題を解く

プログラムの見通しの良さ，性能向上，バグの防止に役立つ

③ 標準ライブラリ

Python では，標準ライブラリとして math, numpy などのパッケージがあり，さまざまな算法（アルゴリズム）を網羅している

④ 算法（アルゴリズム）を駆使しても， 解くのに時間がかかりすぎるため難しいという問題がある

このことは，**暗号**の基礎として役立つ

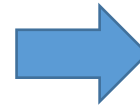
アウトライン

	項目
	復習，全体まとめ
7-1	モジュール，インポート
7-2	標準ライブラリ
7-3	算法（アルゴリズム）

プログラミング (programming)

- コンピュータは，**プログラム**で動く
- プログラミングは，**プログラム**を設計，製作すること
- 何らかの作業を，コンピュータで実行させるために行う

```
a = [200, 400, 300]
for i in a:
    print (i * 1.08)
```



```
216.0
432.0
324.0
```

Python プログラムの**ソースコード**

プログラムの実行結果

ソースコード (source code)

- **プログラム**を，何らかの<u>プログラミング言語</u>で書いたもの
- 「**ソフトウエアの設計図**」ということも.
- <u>**人間も読み書き，編集できる**</u>

```
import picamera
camera = picamera.PiCamera()
camera.capture("1.jpg")
exit()
```

Raspberry Pi で，カメラを使って
撮影し，画像を保存するプログラムの
ソースコード（Python 言語）

Python の変数

- **変数**：**名前の付いたオブジェクト**には，**変数**，**関数**などがある（変数や関数は，数学の変数や関数とは**違う意味**）
- **変数**は，「**値をコンピュータに覚えさせておくもの**」として使うことができる

```
x = 100
print(x)
```

```
100
```

```
y = 2000
print(y)
```

```
2000
```

変数には，値を代入できる

関数

```
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

7-1. モジュール，インポート

ライブラリとモジュール

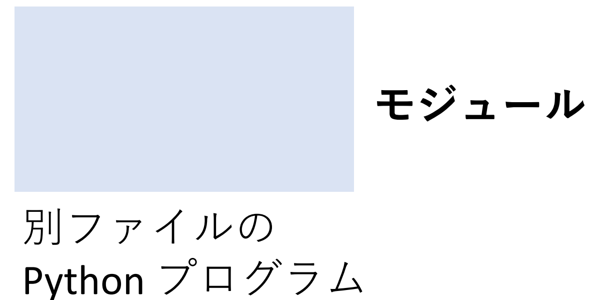
- **ライブラリ**とは
  複数の**プログラム**が共有して使えるような
  機能を持った**プログラム**のこと.
  多くの場合，**プログラム**の実行時に**リンク**（結合）される
- **パッケージ**　（**モジュール**，**インクルードファイル**などともいう）
  複数の**プログラム**が共有して使えるような
  機能を持った**ソースコード**

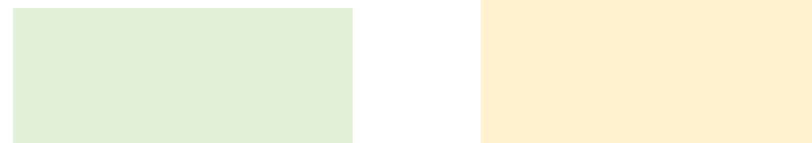
※ パッケージの種類、豊富は、プログラミング言語とに違う

モジュール

Pythonプログラム①　Pythonプログラム②

モジュールを考えない場合





Pythonプログラム①改　Pythonプログラム②改

モジュールを考える場合
（ファイル数は増えるが，
全体として簡潔になる）

モジュール

- **Python のソースコード**が入った**ファイル**を，**モジュール**にすることができる
- **モジュール**は，**インポート可能**（次ページで説明）

```
def foo(x):
    return x + 3
```

モジュール名：bar

ファイルに保存された
Python プログラム
（ファイル名：bar.py)

ファイル名は何でもよいが，それによってモジュール名が決まる

インポート

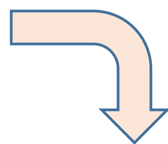
別の**モジュール**を**インポート**することで，**インポート**した**モジュール**の中の **Python ソースコードにアクセスできる**ようになる.

import ＜モジュール名＞

Python でのインポートの方法

```
def foo(x):
    return x + 3
```

ファイルに保存された Python プログラム
（ファイル名：bar.py)



import bar でインポート

```
import bar
print(bar.foo(100))
```

別のPython プログラム

```
>>> import bar
>>> print(bar.foo(100))
103
```

実行結果として 103 が表示される

他のモジュールへのアクセス

1. インポートする

import ＜モジュール名＞

2. モジュールの中の関数等にアクセスする

＜モジュール名＞.＜オブジェクト名＞

「オブジェクト名」は関数名など

```
def foo(x):
    return x + 3
```

ファイルに保存された
Python プログラム
（ファイル名：bar.py)

```
import bar
print(bar.foo(100))
```

別のPython
プログラム

```
>>> import bar
>>> print(bar.foo(100))
103
```

実行結果として
103 が表示される

モジュールのインポートの仕組み

関数 foo

```
def foo(x):
    return x + 3
```

ファイルに保存された
Python プログラム
（ファイル名：bar.py)

import bar でインポート

インポート時に，モジュールを機能させるための変数等が自動で**追加**される

__buildins__, __cached__, __doc__, __file__, __loader__, __name__, __package__, __spec__ を追加

別のPython プログラムと実行結果

```
>>> import bar
>>> print(dir(bar))
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'foo']
>>> _
```

print(dir(bar)) で，モジュール bar のアクセス可能なオブジェクト名（関数名など）が表示される

モジュールの利用例 ①

import hoge でインポート

```
import hoge
print(tax(10))
```

別のPython プログラム

ファイルに保存された Python プログラム （ファイル名：hoge.py)

```
def tax(x):
    return x * 1.08;

if __name__ == "__main__":
    print( tax(100) )
```

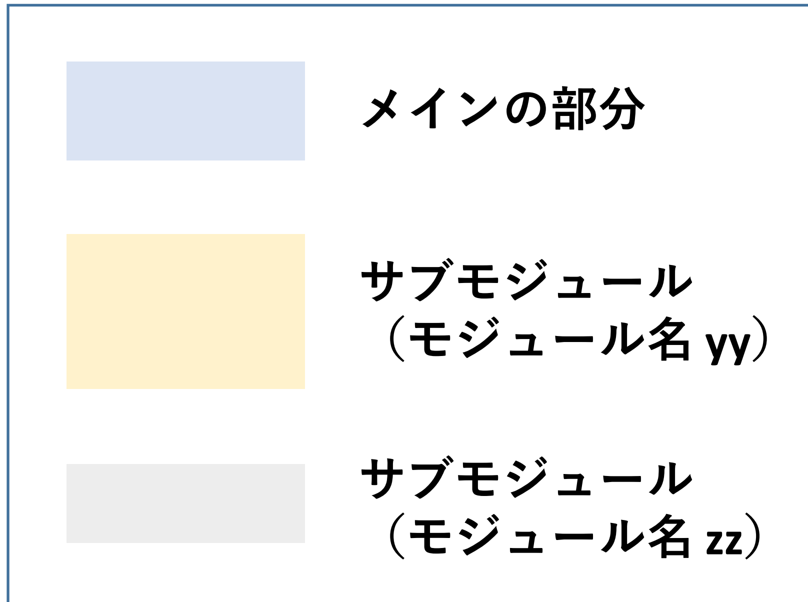
ソースコード

```
kaneko@www:/tmp$ python hoge.py
108.0
```

動作確認結果

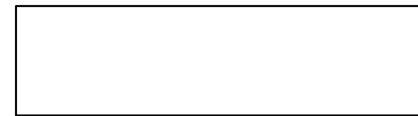
モジュールの利用例 ②

モジュールの機能が大量にあるときなどは，
サブモジュールに分ける場合がある



全体で１つのモジュール
（モジュール名 xx）

インポート
import xx
from xx import yy
from xx import zz
でインポート



別のPython
プログラム

２種類のモジュール

- パッケージ化されていないモジュール

```
def foo(x):
    return x + 3
```

モジュール名 bar

ファイルに保存された
Python プログラム
（ファイル名：bar.py)

- パッケージ化されたモジュール

パッケージ化により，配布，インストールが容易になっている　（詳細割愛）

7-2. 標準ライブラリ

Database Lab.

Python のライブラリ

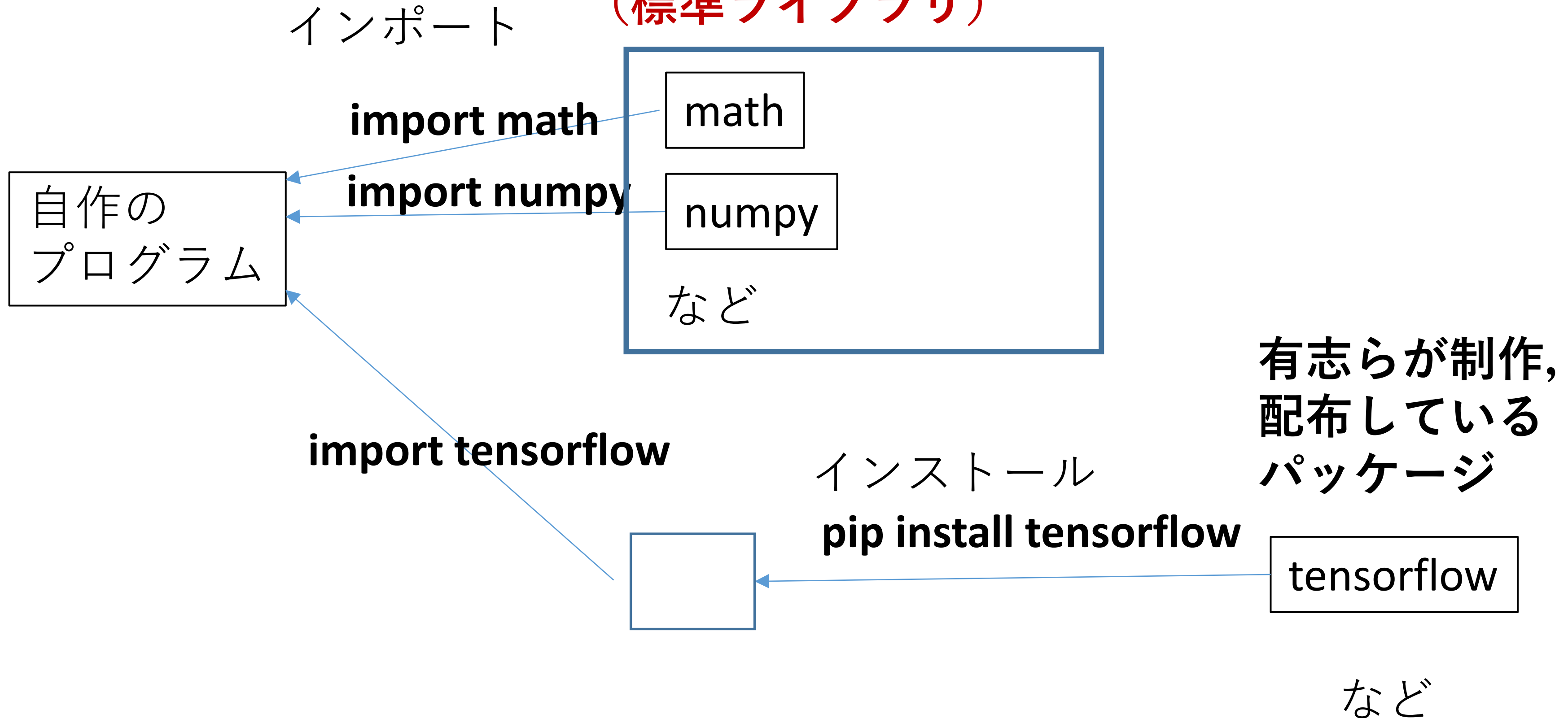
- **標準ライブラリ**の充実．多数の算法（アルゴリズム）が網羅されている．

- **標準ライブラリのほか，多数のパッケージが有志らにより制作，配布されている**（標準ライブラリで足りない場合，補える）

- **オブジェクト指向**の機能を持つ．

  データや関数をオブジェクトとして扱えるだけでなく，モジュールもオブジェクトとして簡単に扱うことができる

標準ライブラリとその他のパッケージ



Python は，パッケージが豊富であることも，人気の理由

Python の標準ライブラリ

- 公式ドキュメント

https://docs.python.org/ja/3/library/index.html

- 組み込み関数，組み込み定数，組み込み型，組み込み例外，テキスト処理，バイナリデータ処理，データ型，数値と数学，関数型プログラミング，ファイルとディレクトリ，データの永続化，デー圧縮とアーカイブ，ファイルフォーマット，暗号，オペレーティングシステム，並列実行，コンテキスト変数，ネットワーク通信とプロセス間通信，インターネット上のデータ操作，HTMLとXML，インターネットプロトコルとサービス，マルチメディアサービス，国際化，プログラムのフレームワーク，グラフィカルユーザインタフェース，開発ツール，デバッグとプロファイル，ソフトウエア・パッケージと配布，Pythonランタイムサービス，カスタム Python インタプリタ，モジュールのインポート，Python 言語サービス，各種サービス

- 多くは「**インポート**」により使用する

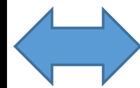
7-3. 算法（アルゴリズム）

算法（アルゴリズム）の例

5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5 × 5　は　**152587890625**

```
In [1]: a = 5 * 5

In [2]: b = a * a

In [3]: c = b * b

In [4]: d = c * c

In [5]: print(d)
152587890625
```

掛け算: 4 回



同じ
結果

```
In [6]: print(5*5*5*5*5*5*5*5*5*5*5*5*5*5*5*5)
152587890625
```

掛け算: 15 回

算法（アルゴリズム）の工夫で，掛け算の回数を削減できる場合がある

次の２つのプログラムは，**同じ答えが出る.**

```
a = 5 * 5
b = a * a
c = b * b
d = c * c
print(d)
```

```
print(5*5*5*5*5*5*5*5*5*5*5*5*5*5*5*5)
```

Print output (drag lower right corner to resi

```
152587890625
152587890625
```

Frames Obj

Global frame

a	25
b	625
c	390625
d	152587890625

次の２つのプログラムは，**同じ答えが出る**

```
a = 10 * 10
b = a * a
c = b * b
print(c)
```

```
print(10*10*10*10*10*10*10*10)
```

Print output (drag lower right corner to resi

```
100000000
100000000
```

Frames Objects

Global frame

a	100
b	10000
c	100000000

暗号を作成する Python プログラム

In [2]:
```
def DH_exchange(p):
    """ generates a shared DH key """
    g = randrange(1,p-1)
    a = randrange(1,p-1) # Alice's secret
    b = randrange(1,p-1) # Bob's secret
    x = pow(g,a,p)
    y = pow(g,b,p)
    key_A = pow(y,a,p)
    key_B = pow(x,b,p)
    return g, a, b, x, y, key_A, key_B
```

In [3]:
```
DH_exchange(190125101)
```

Out[3]:
```
(138828318, 163869277, 133309977, 103589429, 144268455, 9912066, 9912066)
```

Diffie-Hellman法で暗号化（Pythonを使用）

参考ウェブページ

http://nbviewer.jupyter.org/github/yoavram/CS1001.py/blob/master/recitation4.ipynb

暗号を解読する Python プログラム

```
In [4]: def crack_DH(p, g, x):
            """find secret "a" that satisfies g**a%p == x
            Not feasible for large p"""
            for a in range(1,p-1):
                if a % 100000 == 0:
                    print("Iteration",a) # progress bar
                if pow(g,a,p) == x:
                    return a
            return None
```

```
In [5]: def find_prime(n):
            """ find random n-bit long prime (no leading zeros: 2**(n-1)<= N < 2**n )"""
            while True: #here we're optimistic, but actually we have a good reason to be:
                        # after O(1/n) iterations we expect to find a prime and halt
                candidate = randrange(2**(n - 1), 2**n)
                if is_prime(candidate):
                    return candidate
```

```
In [6]: from math import log
        p = find_prime(10)
        print(p,log(p,2))
        g,a,b,x,y,key_A,key_B = DH_exchange(p)
        print('g',g,'a',a,'b',b,'x',x,'y',y,'key_A',key_A,'key_B',key_B)
        print('a',crack_DH(p,g,x))

        983 9.94104760634058
        g 867 a 598 b 417 x 519 y 75 key_A 393 key_B 393
        a 107
```

Diffie-Hellman法の暗号を解読するプログラム

参考ウェブページ

http://nbviewer.jupyter.org/github/yoavram/CS1001.py/blob/master/recitation4.ipynb

暗号作成より，暗号解読の方が手間がかかる

Database Lab.

- **暗号解読**の**算法（アルゴリズム）**は**発見済み**　ということが多い
  → プログラムを作成可能

- しかし，プログラムが答えを出すまでに**時間がかかる場合**がある

すでに，2012 年に，このようなレポートが.
https://www.dit.co.jp/service/security/report/03.html
英大小文字+数字+記号を組み合わせた ZIP のパスワード

６桁の解読時間：　２分２４秒　＝　超危険
８桁の解読時間：　１４日　＝　危険
１０桁の解読時間：　３４１年

「計算可能性」

- **算法（アルゴリズム）の作成が不可能**という問題は，すでに発見済みである

人間にもコンピュータにも解けない問題

チューリングマシンの停止判定

全体まとめ

① モジュール

プログラムの共通部分を別ファイルにできる機能.

② 算法（アルゴリズム）

※　コンピュータは，ある定まった手順により問題を解くプログラムの見通しの良さ，性能向上，バグの防止に役立つ

③ 標準ライブラリ

Python では，標準ライブラリとして math, numpy などのパッケージがあり，さまざまな算法（アルゴリズム）を網羅している

④ 算法（アルゴリズム）を駆使しても，解くのに時間がかかりすぎるため難しいという問題がある

このことは，**暗号**の基礎として役立つ

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