11 Python基礎

モジュール

- P.59-72までを、良く読んでください

- 標準モジュール / 標準ライブラリ

  > https://docs.python.org/ja/3.9/library/

  - モジュール・パッケージ・ライブラリの違いについて理解する

    > https://pypi.org/

これまでに使用・紹介したモジュール

- 再度、使い方・機能などを確認しておくこと
  - random
  - time
  - math

基本的な使用方法

- 記述の仕方は以下の2通り

```
import <モジュール名>
```

  - 名前空間が汚れないので、基本的にはこちらが推奨される

```
from <モジュール名> import <識別子名>
```

  - <識別子名>：関数・変数・クラス名など
  - 利用時の記述が簡略化できるが、名前の衝突（コンフリクト）に注意が必要

- 別名を利用する場合が多い
  - モジュール名が長いのを、簡単に記述する

【例】

```
import math as m

print(m.pi) # 本来なら math.pi と記述する
```

> https://note.nkmk.me/python-import-usage/

モジュールの検索パス

- Pythonはモジュールをインポートする際、`sys.path`に指定されたディレクトリを順番に検索する
- `sys.path`の内容を確認する方法：

```
import sys
print(sys.path)
```

【実行結果】

```
['', '/usr/lib/python39.zip', '/usr/lib/python3.9', '/usr/lib/python3.9/lib-dynload',
'/home/user/.local/lib/python3.9/site-packages', '/usr/local/lib/python3.9/dist-
packages', '/usr/lib/python3/dist-packages']
```

- 検索パスの順序：
    1. カレントディレクトリ（空文字列`''`で表される）
    2. 標準ライブラリのディレクトリ
    3. サードパーティパッケージのディレクトリ
- 検索パスに新しいディレクトリを追加する方法：

```
import sys
sys.path.append('/path/to/your/module')
```

- 注意点：
    - 検索パスの順序は重要（同じ名前のモジュールがある場合、先に見つかった方が使用される）
    - カレントディレクトリが最初に検索されるため、標準ライブラリと同じ名前のモジュールを作成すると、標準ライブラリが使用できなくなる可能性がある

import , from import 両方の使い方を比べてみる

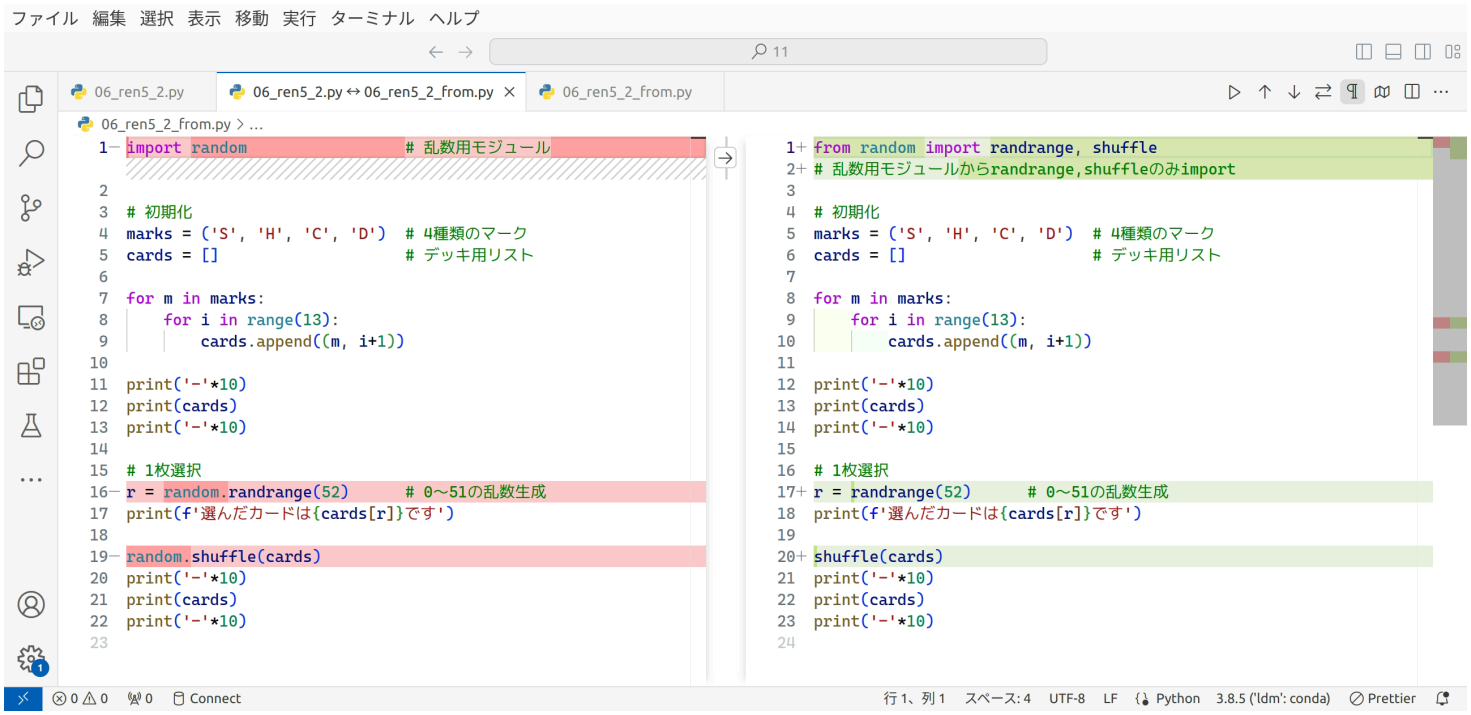
【06_ren5_2.py】

```
import random          # 乱数用モジュール

# 初期化
marks = ('S', 'H', 'C', 'D')  # 4種類のマーク
cards = []                    # デッキ用リスト

for m in marks:
    for i in range(13):
        cards.append((m, i+1))

print('-'*10)
print(cards)
print('-'*10)

# 1枚選択
r = random.randrange(52)       # 0〜51の乱数生成
print(f'選んだカードは{cards[r]}です')

random.shuffle(cards)
print('-'*10)
print(cards)
print('-'*10)
```

- このプログラムを `from import` で書き換えると以下のようになる。
  【06_ren5_2_from.py】

```
from random import randrange,shuffle
# 乱数用モジュールからrandrange,shuffleのみimport

# 初期化
marks = ('S', 'H', 'C', 'D')  # 4種類のマーク
cards = []                    # デッキ用リスト

for m in marks:
    for i in range(13):
        cards.append((m, i+1))

print('-'*10)
print(cards)
print('-'*10)

# 1枚選択
r = randrange(52)       # 0〜51の乱数生成
print(f'選んだカードは{cards[r]}です')
```

```
shuffle(cards)
print('-'*10)
print(cards)
print('-'*10)
```



練習1

- 以下のモジュール・ライブラリを利用したプログラムを作成する
  - 【ren1_1.py】mathモジュールを使用して、$\sqrt{N}$を求める一覧で出力する。N=1～10とする
    【実行結果】

```
√1 = 1.0
√2 = 1.4142135623730951
√3 = 1.7320508075688772
√4 = 2.0
√5 = 2.23606797749979
√6 = 2.449489742783178
√7 = 2.6457513110645907
√8 = 2.8284271247461903
√9 = 3.0
√10 = 3.1622776601683795
```

  - 【ren1_2.py】mathモジュールを利用し、0度～90度（10度刻み）で、`sin() , cos()`の一覧表を出力する
    【実行結果】

```
deg   sin(x)      cos(x)
 0 0.00000000 1.00000000
10 0.17364818 0.98480775
20 0.34202014 0.93969262
30 0.50000000 0.86602540
40 0.64278761 0.76604444
50 0.76604444 0.64278761
60 0.86602540 0.50000000
70 0.93969262 0.34202014
80 0.98480775 0.17364818
90 1.00000000 0.00000000
```

- 【ren1_3.py】datetimeモジュールを使用しする。
  ユーザが入力した日付（YYYY-MM-DD形式）から、現在までの経過日数を出力する
  【実行結果】

```
日付をYYYY-MM-DD形式で入力してください：2024-05-01
経過日数：18日
```

モジュールの作成

- 実際にモジュールを作成し、記述する上での注意事項や、仕組みを理解する

- 以下の関数をモジュールとして作成する
  - 【module_test.py】`add_int()`：2数の和を求め、intで返す

```python
def add_int(num1, num2) -> int:
    return int(num1+num2)
```

  - 通常、テスト用のコードを一緒に記載する。

```python
def add_int(num1, num2) -> int:
    return int(num1+num2)


# main
if __name__ == '__main__':
    print(add_int(1, 2))
    print(add_int(1.0, 2.0))
```

  【実行結果】

```
3
3
```

- モジュールを利用するプログラムを作成する【module_test_main.py】

```
import module_test

print(module_test.add_int(10, 20))
```

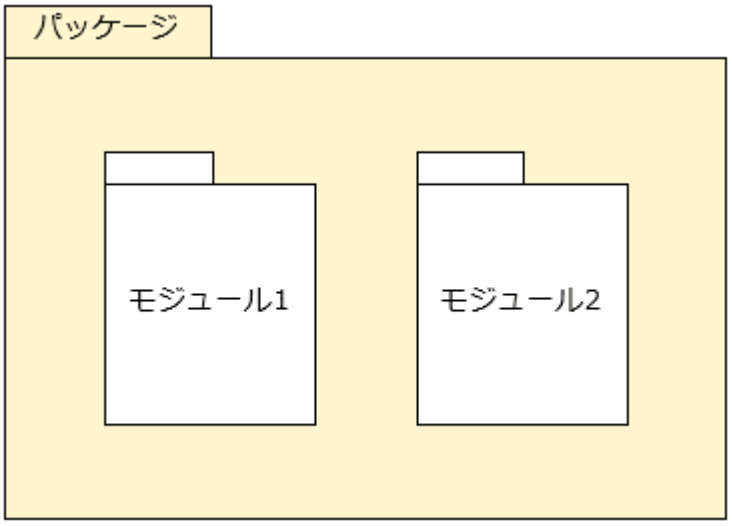
【実行結果】

```
30
```

パッケージの作成

- モジュール内の関数等が増えて、複数のファイルで管理するようになった場合、パッケージ化する

> https://atmarkit.itmedia.co.jp/ait/articles/1907/09/news010.html



パッケージの作成例

- 実際には、フォルダとファイルを利用して、構成する

```
$ tree test_package
test_package
├── __initi__.py
├── mod1.py
└── mod2.py
```

【mod1.py】先に作成した `add_int()`

```
def add_int(num1, num2) -> int:
    return int(num1+num2)


def add_xxx(num1, num2) -> int: # add_intのcopy
    return int(num1+num2)


# __all__ = ['add_int']
# main
if __name__ == '__main__':
    print(add_int(1, 2))
    print(add_int(1.0, 2.0))
```

【mod2.py】 `add_float()`

```
def add_float(num1, num2) -> float:
    return float(num1+num2)


def add_yyy(num1, num2) -> float: # add_floatのcopy
    return float(num1+num2)


# __all__ = ['add_float']
# main
if __name__ == '__main__':
    print(add_float(1, 2))
    print(add_float(1.0, 2.0))
```

【__init__.py】空のファイル

```

```

パッケージの利用例

- この状態で、テストするためのmainを作成する。
  【test_package_main.py】

```
import test_package.mod1
import test_package.mod2

print(test_package.mod1.add_int(10, 20))
print(test_package.mod2.add_float(10, 20))
```

    - 動作するのを確認してください。
- 別の記述方法

【test_package_main_2.py】

```
from test_package import mod1
from test_package import mod2

print(mod1.add_int(10, 20))
print(mod2.add_float(10, 20))
```

- この記述方法だと、少々面倒。
    - そこで、`__init__.py`で設定を行っておく

【__init__.py】

```
from test_package.mod1 import add_int
from test_package.mod2 import add_float

__all__ = ['add_int', 'add_float']
```

【test_package_main_3.py】

```
from test_package import *

print(add_int(10, 20))
print(add_float(10, 20))
```

- 不要な変数、関数を含む場合、コメントにあるように`__all__`を記述することで、必要な関数のみを利用できるように設定可能
    - `__init__.py`に書いても良い

【mod1.py】

```
def add_int(num1, num2) -> int:
    return int(num1+num2)


def add_xxx(num1, num2) -> int:
    return int(num1+num2)


__all__ = ['add_int'] # この場合add_xxx()は利用できない
# main
if __name__ == '__main__':
    print(add_int(1, 2))
    print(add_int(1.0, 2.0))
```

確認

- 対話モードで、import の前後で `dir()` を実行することで、利用可能なものが判る。

```
$ python
Python 3.9.21 (main, Dec 11 2024, 16:24:11)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import time
>>> dir(time)
['CLOCK_BOOTTIME', 'CLOCK_MONOTONIC', 'CLOCK_MONOTONIC_RAW',
'CLOCK_PROCESS_CPUTIME_ID', 'CLOCK_REALTIME', 'CLOCK_TAI', 'CLOCK_THREAD_CPUTIME_ID',
'_STRUCT_TM_ITEMS', '__doc__', '__loader__', '__name__', '__package__', '__spec__',
'altzone', 'asctime', 'clock_getres', 'clock_gettime', 'clock_gettime_ns',
'clock_settime', 'clock_settime_ns', 'ctime', 'daylight', 'get_clock_info', 'gmtime',
'localtime', 'mktime', 'monotonic', 'monotonic_ns', 'perf_counter', 'perf_counter_ns',
'process_time', 'process_time_ns', 'pthread_getcpuclockid', 'sleep', 'strftime',
'strptime', 'struct_time', 'thread_time', 'thread_time_ns', 'time', 'time_ns',
'timezone', 'tzname', 'tzset']
```

練習2

1. フォルダ `ren2_1` を作成し、その中に以下のプログラムを作成してください。
    - `greeting.py` という名前のモジュールを作成し、その中に `say_hello` 関数を定義する
        - この関数は、引数で渡された名前に対して "Hello, [名前]!" と出力する機能を持つ
    - このモジュールを使用してメインスクリプト `main.py` から挨拶を表示する

2. フォルダ `ren2_2` を作成し、その中に以下のプログラムを作成してください。
    - `math_operations.py` という名前のモジュールを作成し、その中に以下の関数を定義する
        - `add(a, b)` - 2つの数値の和を返す
        - `subtract(a, b)` - 2つの数値の差を返す
        - `multiply(a, b)` - 2つの数値の積を返す
        - `divide(a, b)` - 2つの数値の商を返す
    - これらの関数を使って、簡単な計算を行うスクリプト `main.py` を作成する
3. フォルダ `ren2_3` を作成し、その中に以下のプログラムを作成してください。
    - パッケージ `string_operations` を作成し、その中に `reverse_string.py` というモジュールを作成する。
        - `reverse_string.py` の中に `reverse` 関数を定義する
            - この関数は、引数で渡された文字列を逆順にして返す
    - このパッケージを使用したメインスクリプト `main.py` を作成する

練習3

- 以下のモジュールを利用するプログラムを作成したところ、正常に動作しなかった。
    - 原因を突き止めてください。
- 作成したファイル【math.py】

```
from math import sin

x = 45
y = sin(x)
print(y)
```

【実行結果】

```
$ python math.py
Traceback (most recent call last):
  File "/home/yoshimura/src/11/math.py", line 1, in <module>
    from math import sin
  File "/home/yoshimura/src/11/math.py", line 1, in <module>
    from math import sin
ImportError: cannot import name 'sin' from partially initialized module 'math' (most
likely due to a circular import) (/home/yoshimura/src/11/math.py)
```