NOTE

모듈과 패키지

1. 모듈 (Module)

- 파이썬 코드가 들어 있는 파일.
- 코드를 여러 곳에서 재사용하기 쉽게 하는 역할을 함.

2. import 문

- 같은 폴더 안에 있는 py 파일에 있는 함수나 변수를 사용하고 싶을 때 사용.

EX)

```
# main.py
import util

util.print_hello()   # Hello 가 출력됨.
```

```
# util.py
def print_hello():
    print("Hello")
```

- 특정 함수만을 불러 오고 싶을 때는 from 문을 사용.

EX)

```
# main.py
from util import print_hello

print_hello()   # Hello 가 출력됨.
```

- 모든 함수를 불러 오고 싶을 때는 from 문과 별표 (*) 를 사용.

EX)

```
# main.py
from util import *

print_hello()   # Hello 가 출력됨.
```

- 이름을 바꿔 이용하고 싶을 때는 as 문을 사용.

EX)

```
# main.py
from util import print_hello as ph

ph()   # Hello 가 출력됨.
```

NOTE

- import 가 실행되면 해당 모듈이 실행되며,
  해당 모듈에 있는 임포트 대상 함수나 변수의 이름이
  원래 파일의 global 스코프에 담겨짐.

3. 패키지 (Package)

- 여러 모듈이 들어 있는 폴더.
- 마찬가지로 import 문으로 가져올 수 있음.
- 패키지 안의 모듈들은 온점 (.) 으로 구분됨.

EX)

```
# main.py
from util_package import util
util.print_hello()  # Hello 가 출력됨.
```

```
# main.py
from util_package.util import print_hello
print_hello()  # Hello 가 출력됨.
```

- 참고) 별표 (*) 를 활용하면서 몇몇 모듈만을 사용하고 싶을 때는,
  패키지 폴더 안에 __init__.py 파일을 생성한 뒤
  __all__ 변수를 만들어 주면 됨.

EX)

```
# util_package/__init__.py
__all__ = ["module1", "module2"]
```

4. 자주 쓰이는 파이썬 내장 패키지

- 현재 폴더 안에 패키지 폴더가 없더라도 외부 패키지를 임포트하여 사용할 수 있음.

4.1 math 패키지

- 수학에서 많이 사용하는 함수들과 상수들을 정의해 둔 패키지.

- math.sqrt : 제곱근을 반환하는 함수. 항상 실수형으로 반환함.

EX)

```
x = 16
print(math.sqrt(16))  # 4.0
```

- math.pow : 거듭제곱값을 반환하는 함수. 항상 실수형으로 반환함.

EX)

```
x = 4
print(math.pow(x, 2))  # 16.0
```

NOTE

- math.sin , math.cos , math.tan : 삼각함수 계산값을 반환. (라디안)
- math.log : 밑이 자연 상수인 로그 함수의 값을 반환.
  math.log10 : 밑이 10 인 로그 함수의 값을 반환.

- math.pi : 원주율
  math.e : 자연 상수

- math.factorial : 팩토리얼 값을 반환.

EX)
```
print(math.factorial(3))  # 6
```

- math.gcd : 최대공약수 값을 반환.

EX)
```
print(math.gcd(3, 6))  # 3
```

- math.ceil , math.floor : 올림, 내림값을 반환.

EX)
```
print(math.ceil(4.2))  # 5
print(math.floor(4.9))  # 4
```

4.2 random 패키지

- 무작위적인 선택을 가능하게 해주는 패키지.
- random.random : 0.0 과 1.0 사이의 임의의 실수를 반환.
- random.randint : 주어진 두 정수 사이의 정수 중 임의의 정수를 반환. (양 끝 값 포함.)
- random.uniform : 주어진 두 수 사이의 임의의 실수를 반환.
- random.choice : 주어진 리스트나 튜플에서 임의의 요소를 반환.

EX)
```
my_list = [1, 2, 3, 4, 5]
print(random.choice(my_list))
```

- random.choices : 중복 허용하여 여러 개의 요소를 선택. (가중치 부여 가능)
  random.sample : 중복 허용하지 않고 여러 개의 요소를 선택.

EX)
```
my_list = [1, 2, 3, 4, 5]
element_weight = [0.2, 0.1, 0.3, 0.2, 0.2]
print(random.choices(my_list, k=3, weights=element_weight))
print(random.sample(my_list, k=3))
```

- random.shuffle : 리스트나 튜플을 재배열.

EX)

NOTE

```
my_list = [1, 2, 3, 4, 5]
random.shuffle(my_list)
print(my_list)
```

4.3 os 패키지

- 운영 체제에서 제공하는 기능을 제어하고, 파일과 디렉토리 작업을 수행하는 패키지.
- os.getcwd : 현재 작업 디렉토리를 반환하는 함수.

EX)
```
print(os.getcwd())  # 현재 디렉토리 경로 출력
```

- os.listdir : 지정된 디렉토리의 파일 및 폴더 목록을 반환하는 함수.

EX)
```
print(os.listdir('.'))  # 현재 디렉토리의 파일 목록 출력
```

- os.mkdir : 새로운 디렉토리를 생성하는 함수.

EX)
```
os.mkdir('new_folder')  # 'new_folder' 디렉토리 생성
```

- os.rmdir : 빈 디렉토리를 삭제하는 함수.

EX)
```
os.rmdir('new_folder')  # 'new_folder' 디렉토리 삭제
```

- os.remove : 파일을 삭제하는 함수.

EX)
```
os.remove('example.txt')  # 파일 삭제
```

4.4 datetime 패키지

- 날짜와 시간을 처리하는 다양한 기능을 제공하는 패키지.
- datetime.datetime.now : 현재 날짜와 시간을 반환하는 함수.

EX)
```
print(datetime.datetime.now())  # 현재 시간 출력
```

- datetime.timedelta : 두 날짜 또는 시간의 차이를 나타내는 객체.

EX)
```
delta = datetime.timedelta(days=1)
print(datetime.datetime.now() + delta)  # 하루 뒤 시간 출력
```

- datetime.datetime.strptime : 문자열을 지정된 형식으로 날짜 객체로 변환하는 함수.

EX)
```
date = datetime.datetime.strptime('2024-10-12', '%Y-%m-%d')
print(date)  # '2024-10-12 00:00:00' 출력
```

- datetime.datetime.strftime : 날짜 객체를 지정된 형식의 문자열로 변환하는 함수.

EX)

NOTE

```
now = datetime.datetime.now()
print(now.strftime('%Y-%m-%d %H:%M:%S'))  # 날짜를 문자열로 변환
```

- datetime.datetime.date : 날짜 부분만 반환하는 함수.
  datetime.datetime.time: 시간 부분만 반환하는 함수.

  EX)
```
print(datetime.datetime.now().date())
# '2024-10-12' 같은 형태의 날짜 반환
print(datetime.datetime.now().time())
# '14:30:00' 같은 형태의 시간 반환
```

- datetime.datetime.combine: 날짜와 시간을 결합하여 새로운 datetime 객체를 생성하는 함수.

  EX)
```
date = datetime.date(2024, 10, 12)
time = datetime.time(14, 30)
combined = datetime.datetime.combine(date, time)
print(combined)  # '2024-10-12 14:30:00' 출력
```

4.5 collections 패키지

- 고급 자료 구조를 제공하는 패키지로, 기본 파이썬 자료 구조의 확장형을 지원.
- collections.Counter: 요소의 개수를 세는 딕셔너리 형태의 객체.

  EX)
```
from collections import Counter
c = Counter(['a', 'b', 'a', 'c', 'b', 'b'])
print(c)  # {'b': 3, 'a': 2, 'c': 1}
```

- collections.defaultdict: 기본 값을 설정할 수 있는 딕셔너리 객체.

  EX)
```
from collections import defaultdict
d = defaultdict(int)
d['a'] += 1
print(d)  # {'a': 1}
```

- collections.namedtuple: 튜플을 이름이 있는 필드로 확장하여 사용.

  EX)
```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
p = Point(11, 22)
print(p.x, p.y)  # 11 22 출력
```

- collections.deque: 양방향 큐를 제공하는 자료 구조로, 빠른 삽입 및 삭제가 가능.

  EX)

NOTE

```
from collections import deque
d = deque([1, 2, 3])
d.append(4)
d.appendleft(0)
print(d)  # deque([0, 1, 2, 3, 4])
```

- collections.OrderedDict: 삽입 순서를 기억하는 딕셔너리.

EX)
```
from collections import OrderedDict
od = OrderedDict()
od['a'] = 1
od['b'] = 2
print(od)  # OrderedDict([('a', 1), ('b', 2)])
```

- collections.ChainMap: 여러 딕셔너리를 하나의 뷰로 결합.

EX)
```
from collections import ChainMap
d1 = {'a': 1, 'b': 2}
d2 = {'c': 3, 'd': 4}
cm = ChainMap(d1, d2)
print(cm)  # ChainMap({'a': 1, 'b': 2}, {'c': 3, 'd': 4})
```

4.6 itertools 패키지

- 반복자(Iterator) 생성과 관련된 다양한 함수들을 제공하는 패키지.
- itertools.chain: 여러 반복 가능한 객체를 하나의 시퀀스로 결합.

EX)
```
from itertools import chain
print(list(chain([1, 2], [3, 4])))  # [1, 2, 3, 4]
```

- itertools.combinations: 주어진 리스트에서 가능한 조합을 반환.

EX)
```
from itertools import combinations
print(list(combinations([1, 2, 3], 2)))  # [(1, 2), (1, 3), (2, 3)]
```

- itertools.permutations: 주어진 리스트에서 가능한 순열을 반환.

EX)
```
from itertools import permutations
print(list(permutations([1, 2, 3], 2)))
# [(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]
```

- itertools.product: 두 리스트의 데카르트 곱을 반환.

EX)

NOTE

```
from itertools import product
print(list(product([1, 2], ['a', 'b'])))  # [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b')]
```

- itertools.cycle: 리스트의 요소들을 무한 반복하는 반복자 생성.

EX)
```
from itertools import cycle
cy = cycle([1, 2, 3])
for _ in range(5):
    print(next(cy), end=' ')  # 1 2 3 1 2 출력
```

- itertools.count: 지정한 값부터 무한히 증가하는 반복자 생성.

EX)
```
from itertools import count
for i in count(10, 2):
    if i > 15:
        break
    print(i, end=' ')  # 10 12 14 출력
```

- itertools.groupby: 리스트의 연속된 동일한 값을 그룹으로 묶음.

EX)
```
from itertools import groupby
print([(k, list(g)) for k, g in groupby('AAAABBBCCD')])
# [('A', ['A', 'A', 'A', 'A']), ('B', ['B', 'B', 'B']), ('C', ['C', 'C']), ('D', ['D'])]
```