파이썬 입문

한국폴리텍대학

2023.05.26

module 만들기 1

```
# test.py

import mod
# import mod as md
# from mod import *
# from mod import a, add, calc

print(mod.a)                  # 100
print(mod.add(3, 2))      # 5

m = mod.calc()
print(m.mul(3, 2))          # 6
```

```
# mod.py

a = 100

def add(x1, x2):
        return x1 + x2

class calc():
        def mul(self, x1, x2):
                return x1 * x2

# print(“module”)
```

module 만들기 2

```
# test.py

from mod1 import *
from mod2 import *

print(a)                  # 100
print(add(3, 2))      # 5

m = calc()
print(m.mul(3, 2))          # 6
```

```
# mod1.py

a = 100

def add(x1, x2):
        return x1 + x2

class calc():
        def mul(self, x1, x2):
                return x1 * x2

# print("module1")
```

```
# mod2.py

a = 200

def add(x1, x2):
        return x1 + x2 + 100

class calc():
        def mul(self, x1, x2):
                return x1 * x2 + 100

# print("module2")
```

package 만들기

```
# test.py

import pkg.mod
# import pkg.mod as md
# from pkg.mod import *
# from pkg.mod import add

print(pkg.mod.a)                    # 100
print(pkg.mod.add(3, 2))      # 5

m = pkg.mod.calc()
print(m.mul(3, 2))                # 6

# print(__name__)                   # __main__
# print(pkg.mod.__name__)    # pkg.mod
```

```
# pkg 폴더 생성
# pkg 폴더 안에 __init__.py 파일 생성
# pkg 폴더 안에 mod.py 파일 생성

# __init__.py          # python 3.3 이상부터 없어도 됨
# 내용 없음

# mod,py
a = 100

def add(x1, x2):
        return x1 + x2

class calc():
        def mul(self, x1, x2):
                return x1 * x2

If __name__ == '__main__':
        print("main")
else:
        print("package module")
```

쓰레드 (Thread) 1

```
import threading
import time

def worker1(x1, x2):
	while x2 > 0:
		print(f‘sub thread: {x1} [{x2}]₩n’)
		time.sleep(1)
		x2 -= 1

def worker2(x1, x2):
	while x2 > 0:
		print(f‘sub thread: {x1} [{x2}]₩n’)
		time.sleep(1)
		x2 -= 1

t1 = threading.Thread(target = worker1, args = (“designer”, 20))
t2 = threading.Thread(target = worker2, args = (“programmer”, 10))
t1.start()
t2.start()

print(‘main thread’)
```

```
# process
# thread
```

쓰레드 (Thread) 2

```
import threading
import time

def worker1(x1, x2):
        while x2 > 0:
                print(f‘sub thread: {x1} [{x2}]₩n’)
                time.sleep(1)
                x2 -= 1

def worker2(x1, x2):
        while x2 > 0:
                print(f‘sub thread: {x1} [{x2}]₩n’)
                time.sleep(1)
                x2 -= 1

t1 = threading.Thread(target = worker1, args = (“designer”, 20))
t2 = threading.Thread(target = worker2, args = (“programmer”, 10))
t1.daemon = True
t1.start()
t2.daemon = True
t2.start()

print(‘main thread’)
```

# daemon : main thread가 종료되면 같이 종료함

쓰레드 (Thread) 3

```
import threading
import time

def worker1(x1, x2):
        while x2 > 0:
                print(f‘sub thread: {x1} [{x2}]₩n’)
                time.sleep(1)
                x2 -= 1

def worker2(x1, x2):
        while x2 > 0:
                print(f‘sub thread: {x1} [{x2}]₩n’)
                time.sleep(1)
                x2 -= 1

t1 = threading.Thread(target = worker1, args = (“designer”, 20))
t2 = threading.Thread(target = worker2, args = (“programmer”, 10))
t1.join()
t1.start()
t2.join()
t2.start()

print(‘main thread’)
```

```
# join : main thread는 sub thread가
         종료될 때까지 기다림
```

쓰레드 (Thread) 4

```
import threading
import time
num = 0
lock = threading.Lock()
# lock = threading.Semaphore(1)
print(type(lock))

def worker1(x1, x2):
      global num
      while x2 > 0:
            print(f‘sub thread: {x1} [{x2}]₩n’)
            lock.acquire()
            time.sleep(1)
            x2 -= 1 ; num += 1
            print(f'num1 : [{num}]₩n')
            lock.release()

def worker2(x1, x2):
      global num
      while x2 > 0:
            print(f‘sub thread: {x1} [{x2}]₩n’)
            lock.acquire()
            time.sleep(1)
            x2 -= 1; num += 1
            print(f’num2 : [{num}]₩n')
            lock.release()

t1 = threading.Thread(target = worker1, args = (“designer”, 20))
t2 = threading.Thread(target = worker2, args = (“programmer”, 10))
t1.start()
t2.start()
print(‘main thread’)
```

# 임계영역 (critical section)
# 뮤텍스 (Mutex (Mutual Exclusion))
   Lock --> acquire, release

# 세마포어 (Semaphore)

## Cython

# Cython은 CPython 패키지를 만들 수 있는 라이브러리

# Cython으로 작성된 파일의 확장자는 .pyx

1. pip install cython   # 파이썬이 여러 버전 설치되어 있는 경우 pip3, pip3.11

2. test.pyx 만들기

```
m = 0
def loop_test(n):
        for i in range(n):
                for j in range(n):
                        m += 1
        return m
```

3. setup.py 만들기

```
# -*- coding: utf-8 -*-
from distutils.core import setup
from Cython.Build import cythonize
setup(ext_modules=cythonize( " test.pyx " )) # 어떤 파일을 변환할지 지정
```

4. Build --> test.c , test.cp311-win_amd64.pyd 생성

```
python setup.py build_ext –inplace
```

# running build_ext : .pyx 내용을 바꾼 후, build_ext 재실행

5. test2.py 사용하기

```
import test   # CPython 패키지
import time

start = time.time()
test.loop_test(5000)
end = time.time()
print(f"{end - start:.6f} sec.")   # 1.200000 sec.

def loop_test(n:int)->int:
    m = 0
    for i in range(n):
        for j in range(n):
            m += 1
    return m

start = time.time()
loop_test(5000)
end = time.time()
print(f"{end - start:.6f} sec.")   # 12.000000 sec.
```

# CPython 패키지가 빠르다.