파이썬 프로그래밍
강의노트 #13

모듈활용

모듈(Module)

- 모듈
  - 관련된 상수, 함수 또는 클래스 등을 모아놓은 파일
- 모듈의 필요성
  - 코드 관리의 용이성
    - 코드가 길어지면 한 개 파일에 모든 코드를 넣고 관리하는 것이 어려움
  - 실행의 효율성
    - 긴 코드를 메모리에 올려서 실행시키는 것은 비효율적
  - 재사용성
    - 관련된 함수나 클래스들을 한 개 파일에 묶으면 재사용이 용이해짐

모듈 만들기

- 파이썬 함수나 클래스를 작성하고 파일에 저장
- 강의노트 12에서 만들었던 Rectangle과 Circle 클래스를 Rectangle.py와 Circle.py에 저장

```
# Rectangle.py
class Rectangle:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
    def calcArea(self):
        return (self.x2 - self.x1) * (self.y1 - self.y2)
```

모듈 만들기

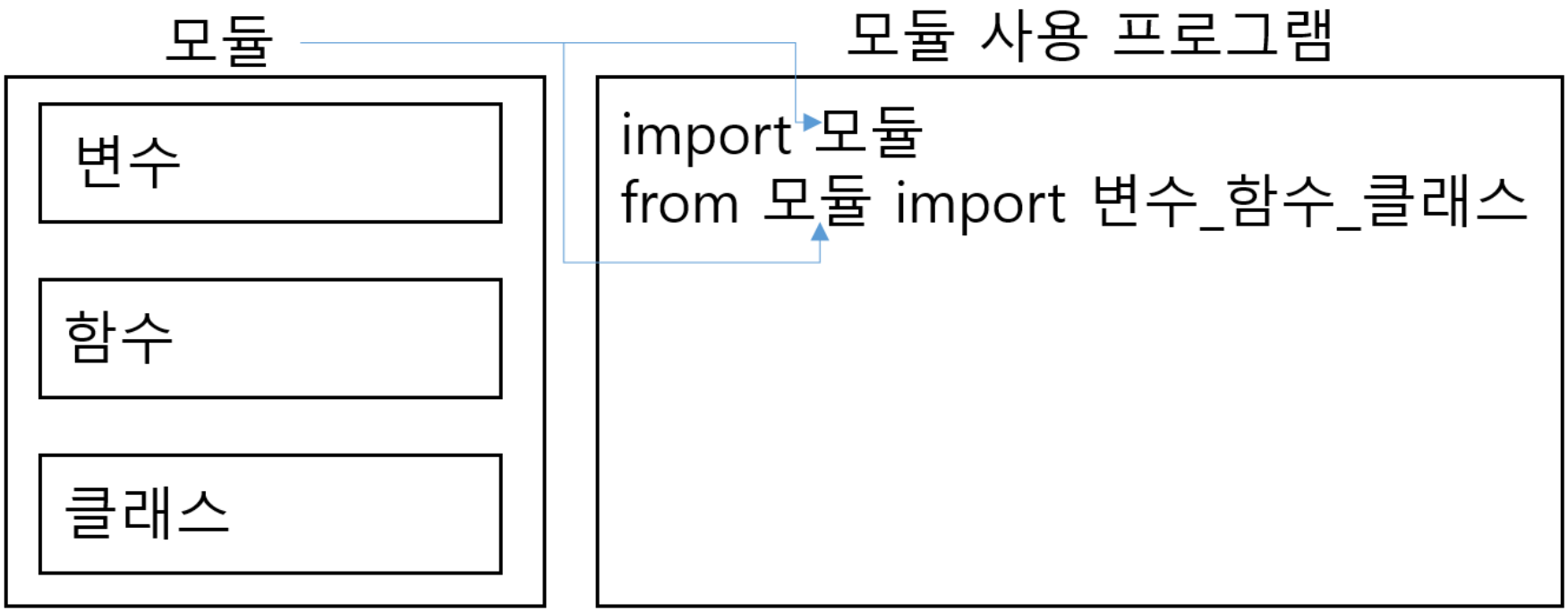
```
# Circle.py
import math

class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    def calcArea(self):
        return math.pi * self.r * self.r
```

모듈 사용법

- 다른 모듈 파일을 사용하려면 import해야 함
  - 모듈_이름.함수_이름 또는 모듈_이름.클래스_이름 형태로 사용 가능
  - from 모듈_이름 import 함수_이름 형태로 사용 가능



모듈 사용법

- Rectangle과 Circle 모듈 사용 예

```
>>> import Rectangle
>>> import Circle
```

- 여러 모듈을 한꺼번에 import 가능

```
>>> import Rectangle, Circle
```

- 모듈의 변수, 클래스나 함수 등을 사용

```
>>> r = Rectangle.Rectangle(10, 20, 20, 10)
>>> print(r.calcArea())
100
>>> c = Circle.Circle(10, 10, 20)
>>> print(c.calcArea())
1256.6370614359173
```

함수만 있는 모듈 만들기

- 주사위 1~6 사이의 숫자를 생성하는 모듈

```
# dice1.py
import random

def throwDice():
    return random.randint(1, 6)
```

- dice1 사용 코드

```
>>> import dice1
>>> print(dice1.throwDice())
2
>>> print(dice1.throwDice())
6
```

모듈에 변수 넣고 수정해보기

- 모듈에 변수를 넣고 수정하는 것도 가능
- 범위를 지정할 수 있는 주사위 모듈 구현

```
# dice2.py
import random

minNum = 1
maxNum = 10

def throwDice():
    return random.randint(minNum, maxNum)
```

모듈에 변수 넣고 수정해보기

- dice2 사용 코드

```
>>> import dice2
>>> print(dice2.throwDice())
8
>>> print(dice2.throwDice())
1
>>> print(dice2.throwDice())
4
```

모듈에 변수 넣고 수정해보기

- 변수값 수정 후 사용

```
>>> dice2.minNum = 11
>>> dice2.maxNum = 20
>>> print(dice2.throwDice())
11
>>> print(dice2.throwDice())
12
>>> print(dice2.throwDice())
15
```

from...import로 모듈 이름 줄이기

- from ... import를 사용하면 매번 모듈 이름을 입력하지 않아도 됨

```
from 모듈_이름 import 변수_함수_클래스_이름
```

  - import된 것을 사용할 때 이름만 사용해도 됨

- Rectangle 클래스 사용

```
>>> from Rectangle import Rectangle
>>> r = Rectangle(10, 30, 20, 10)
>>> print(r.calcArea())
200
```

- dice1 사용

```
>>> from dice1 import throwDice
>>> print(throwDice())
1
```

실습문제 1

- 문제
  - 다음 내용을 클래스 외부에 정의
    - 원주율 pi 변수(3.1415)
    - 원의 둘레를 계산하는 circumference(인자 r을 받음) 함수
  - 원 클래스를 구현
    - 원의 생성자는 반지름을 인자로 전달받고 멤버 변수에 저장
    - 멤버 함수로는 원주를 반환하는 getCircumference()를 구현

실습문제 1

- 요구사항
  - Circle 클래스의 getCircumference() 함수는 모듈 내에 있는 circumference() 함수를 사용
  - 원주율은 Circle1모듈의 내부 변수인 pi를 사용
  - prac13_01.py에서는 Circle1 모듈의 pi값을 화면에 출력하고, circumference() 함수와 Circle 클래스의 getCircumference() 함수의 결과값을 화면에 출력
  - pi, circumference(), Circle 클래스를 Circle1 모듈에 넣을 것

실습문제 1

- 최종 코드

```
# Circle1.py
pi = 3.1415

def circumference(r):
    return 2 * pi * r

class Circle:
    def __init__(self, r):
        self.r = r

    def getCircumference(self):
        return circumference(self.r)
```

실습문제 1

- 최종 코드

```
# prac13_01.py
import Circle1

print("Circle1.pi 값 출력")
print(Circle1.pi)
print("반지름이 10인 원주 계산.
Circle1.circumference() 함수 사용")
print(Circle1.circumference(10))
print("반지름이 10인 원주 계산. Circle1.Circle 객체
생성 후 getCircumference() 함수 사용")
c = Circle1.Circle(10)
print(c.getCircumference())
```

__name__변수 활용

- 모듈을 만들면서 검수 코드를 넣는 경우가 있음

```
# dice3.py
import random

def throwDice():
    return random.randint(1, 6)

print("testing throwDice()")
for i in range(10):
    print(throwDice(), end = ' ')
print("")
```

__name__변수 활용

- dice3모듈을 사용해서 1~6 사이의 정수를 출력하는 프로그램 TestDice3.py를 작성하고 실행

```
# TestDice3.py
import dice3

print("Testing dice3 module")
for i in range(3):
    print(dice3.throwDice())
```

- 실행 결과

```
testing throwDice()
3 2 1 4 5 1 4 1 2 1
Testing dice3 module
1
4
2
```

dice3.py의 검수 코드가 실행된 후, TestDice3.py의 코드가 실행됨

__name__변수 활용

- 직접 모듈 파일을 실행시키는 것이 아니면 검수 코드가 실행되지 않도록 만들고 싶음
- __name__변수는 파이썬에 내장된 변수
- 파이썬은 코드가 직접 실행될 때 __name__변수를 "__main__"으로 지정
- 모듈이 import되면 __name__변수값이 "__main__"이 아님

__name__변수 활용

```
# dice4.py
import random

def throwDice():
    return random.randint(1, 6)
print(f"dice4: __name__ = {__name__}")
print("testing throwDice()")
for i in range(10):
    print(throwDice(), end = ' ')
print("")
```

- 실행 결과(dice4.py를 직접 실행)

```
dice4: __name__ = __main__
testing throwDice()
1 2 3 3 5 4 6 2 2 6
```

__name__변수 활용

- TestDice4.py를 실행

```
# TestDice4.py
import dice4

print("Testing dice4 module")
print(f"TestDice4: __name__ = {__name__}")
for i in range(3):
    print(dice4.throwDice())
```

- 실행 결과

```
dice4: __name__ = dice4
testing throwDice()
4 3 6 5 2 6 6 2 3 3
Testing dice4 module
TestDice4: __name__ = __main__
5
6
6
```

__name__변수 활용

- __name__변수를 사용해서 dice모듈 수정

```
# dice5.py
import random

def throwDice():
    return random.randint(1, 6)

if __name__ == "__main__":
    print("testing throwDice()")
    for i in range(10):
        print(throwDice(), end = ' ')
    print("")
```

__name__변수 활용

- dice5를 사용하는 코드 작성

```
# TestDice5.py
import dice5

print("Testing dice5 module")
print(f"TestDice5: __name__ = {__name__}")
for i in range(3):
    print(dice5.throwDice())
```

- 실행 결과

```
Testing dice5 module
TestDice5: __name__ = __main__
1
4
6
```

패키지, 라이브러리, 패키지 관리 프로그램

- 패키지(package)
  - 관련된 모듈들을 묶어놓은 것
- 라이브러리
  - 관련된 패키지들을 모아서 한 개의 묶음으로 만든 것
- 패키지 관리 프로그램
  - 다른 사람들이 만들어둔 패키지(모듈들)을 쉽게 설치하고 사용할 수 있도록 pip라는 관리 도구를 제공
- pip 사용 방법
  - 때로는 pip 대신 pip3를 사용하기도 함

```
pip install 패키지_이름
pip3 install 패키지_이름
```

```
pip uninstall 패키지_이름
pip3 uninstall 패키지_이름
```

날짜와 시간 다루기

- date 클래스
  - 현재 날짜를 찾거나, 날짜를 지정해서 요일을 확인하거나, 지정된 서식으로 년월일 정보를 문자열로 변환 가능
  - 오늘 날짜 정보를 취득하려면 today() 함수 사용

```
>>> from datetime import date
>>> d = date.today()
>>> print(d)
2022-08-18
```

  - date 클래스의 weekday() 함수는 요일을 정수로 반환
    - 월요일이 0, 화요일이 1, …

```
>>> print(d.weekday())
3
```

날짜와 시간 다루기

- date 객체를 생성하면서 년월일을 지정할 수 있음

```
>>> d2 = date(2021, 8, 18)
>>> print(d2.weekday())
2
```

- date.strftime()함수는 날짜 정보를 사용자가 지정한 서식의 문자열로 만드는 함수

```
>>> s = d.strftime("%Y/%m/%d")
>>> print(s)
2022/08/18
>>> print(d.strftime("%Y %m %d, 요일: %a"))
2022 08 18, 요일: Thu
>>> print(d.strftime("%Y년 %m월 %d일 요일: %A"))
2022년 08월 18일 요일: Thursday
```

날짜와 시간 다루기

- time 클래스
  - datetime 모듈의 time 클래스는 일반적인 시간 정보를 나타냄
  - 특정 시각 정보를 나타내지 않음(time 모듈과 다름)
  - 주로 datetime 클래스의 멤버 함수의 반환값 또는 시간 정보를 서식에 맞춰 문자열로 생성할 때 사용됨
  - 생성자는 시/분/초/마이크로초/시간대 정보를 인자로 전달 받음(생략 가능)
  - 12시 35분 25초를 객체로 생성

```
>>> from datetime import time
>>> t = time(12, 35, 25)
>>> print(t)
12:35:25
```

날짜와 시간 다루기

- 서식에 맞춰 출력

```
>>> print(t.strftime("%H %M %S"))
12 35 25
>>> print(t.strftime("%H시 %M분 %S초"))
12시 35분 25초
```

- time 모듈
  - 정해진 기준(epoch)를 중심으로 시간 정보를 반환
    - 기준은 1970년 1월 1일 0시 0분 0초
  - time모듈의 gmtime()이나 localtime()함수는 각각 UTC와 지역 시간대의 현재 시각 정보를 일종의 튜플인 struct_time 형태로 반환

날짜와 시간 다루기

- time 모듈 사용

```
>>> import time
>>> ut = time.gmtime()
>>> print(ut)
time.struct_time(tm_year=2022, tm_mon=8, tm_mday=18,
tm_hour=4, tm_min=3, tm_sec=41, tm_
wday=3, tm_yday=230, tm_isdst=0)
>>> lt = time.localtime()
>>> print(lt)
time.struct_time(tm_year=2022, tm_mon=8, tm_mday=18,
tm_hour=13, tm_min=5, tm_sec=18, tm_
wday=3, tm_yday=230, tm_isdst=0)
>>> print(lt.tm_year)
2022
```

날짜와 시간 다루기

- time.time() 함수는 정해진 기준(epoch)이후 몇 초가 흘렀는지 반환
  - 실행 시간 등을 확인할 때 사용됨

```
import time

t1 = time.time()
sum = 0
maxNum = 10000000
for i in range(1, maxNum):
    sum += i
t2 = time.time()
print(f"1부터 {maxNum} 까지의 합을 구하는데 걸리는 시간: {t2 - t1}")
```

경로와 파일 관리 및 파일 목록 뽑기

- 현재 경로 파악 또는 변경
  - os.getcwd() 함수는 현재 디렉토리를 문자열로 반환

```
>>> import os
>>> print(os.getcwd())
C:\
```

  - chdir(path) 함수를 사용하면 원하는 경로로 현재 작업 디렉토리를 변경

```
>>> os.chdir("C:\\temp")
>>> print(os.getcwd())
C:\temp
```

경로와 파일 관리 및 파일 목록 뽑기

- 경로 생성 또는 삭제
  - mkdir() – 생성
  - rmdir() – 삭제

```
>>> os.mkdir("C:\\temp\\new") # 디렉토리 생성
>>> print(os.path.exists("C:\\temp\\new"))
True
>>> os.rmdir("C:\\temp\\new") # 디렉토리 삭제
>>> print(os.path.exists("C:\\temp\\new"))
False
```

경로와 파일 관리 및 파일 목록 뽑기

- 경로 또는 파일 존재 확인

```
>>> os.path.exists("C:\\temp\\test\Test1.class")
True
>>> os.path.exists("C:\\temp\\test\Test1.txt")
False
```

- 파일 및 서브 디렉토리 목록 추출
  - listdir(path) 함수는 주어진 경로 path에 있는 파일과 서브 디렉토리 목록을 리스트로 반환

```
>>> fileList = os.listdir("C:\\temp\\test")
>>> for f in fileList:
... print(f)
...
test.txt
Test1.class
Test1.java
```

경로와 파일 관리 및 파일 목록 뽑기

- 주어진 경로가 파일인지 디렉토리인지 확인
  - isfile(path) – 파일인지 확인
  - isdir(path) – 디렉토리인지 확인

```
>>> os.path.isdir("C:\\Temp")
True
>>> os.path.isfile("C:\\Temp\\test.txt")
True
>>> os.path.isfile("C:\\Temp")
False
```

- 파일 크기 확인
  - os.path.getsize(filepath)는 파일 크기를 바이트로 반환

```
>>> os.path.getsize("C:\\Temp\\Test1.java")
158
```

경로와 파일 관리 및 파일 목록 뽑기

- 파일 삭제
  - os.remove(filepath)

```
# 현재 작업 디렉토리에서 test.txt를 삭제
>>> os.remove('test.txt')
```

- 와일드 카드를 이용한 파일 목록 뽑기
  - 자주 사용되는 와일드 카드 두 가지

와일드 카드	설명
?	문자열에서 ?가 있는 위치의 한 글자 매칭
*	문자열에서 *는 0개 이상의 글자에 매칭

  - 와일드 카드를 이용해서 파일 이름 매칭 기능을 사용하려면 glob 모듈 사용

경로와 파일 관리 및 파일 목록 뽑기

- C:₩Temp₩test 폴더에 있는 파일과 서브디렉토리 목록을 뽑아보기

```
>>> import os
>>> dir = "C:\\Temp\\test"
>>> fileLst1 = os.listdir(dir)
>>> print(fileLst1)
['subtest', 'test.py', 'Test1.class', 'Test1.java',
'test2.txt', 'test3.txt', 'test4.txt',
'test5.py']
```

경로와 파일 관리 및 파일 목록 뽑기

- glob.glob()을 사용해서 모든 파일 목록 뽑기

```
>>> import glob
>>> dir = "C:\\Temp\\test"
>>> print(glob.glob(dir + "\\*"))
['C:\\Temp\\test\\subtest',
'C:\\Temp\\test\\test.py',
'C:\\Temp\\test\\Test1.class',
'C:\\Temp\\test\\Test1.java', 'C:\
\Temp\\test\\test2.txt',
'C:\\Temp\\test\\test3.txt',
'C:\\Temp\\test\\test4.txt', 'C:\\
Temp\\test\\test5.py']
```

경로와 파일 관리 및 파일 목록 뽑기

- glob.glob()을 사용해서 "test"로 시작하고 ".py"로 끝나며, "test"와 "." 사이에 한 글자가 들어있는 파일 목록 추출

```
>>> print(glob.glob("test?.py"))
['test5.py']
```

명령행 인자 받기

- 명령행 인자(command line arguments)란 파이썬 프로그램을 실행시키면서 사용자가 추가로 입력하는 값들
  - 다음 실행문에서 CopyFile.py, abc.txt나 abcd.txt가 명령행 인자

```
C:\temp>python CopyFile.py abc.txt abcd.txt
```

  - 명령행 인자를 확인하려면 sys모듈을 import하고 sys.argv 변수를 사용해야 함
    - sys.argv에는 명령행 인자들이 리스트의 요소로 있음

```
import sys
print(sys.argv) # sys.argv를 출력
print(f"len(sys.argv) = {len(sys.argv)}")
for argument in sys.argv:
    print(argument)
```

실습문제 1

- 문제
  - 와일드 카드의 파일 이름을 명령행 인자로 요청하면, 현재 작업 디렉토리에서 매칭되는 파일의 목록을 화면에 출력
- 요구사항
  - 요청한 파일이 없으면 모든 파일을 출력
  - 매칭되는 파일이 없으면 "매칭되는 파일이 없습니다"를 출력
  - 현재 작업 디렉토리는 소스 코드가 있는 디렉토리로 가정

실습문제 1

- 최종 코드

```
import glob, sys

lst = [] # 파일 목록
if len(sys.argv) == 1: # 파이썬 코드만 있음
    lst = glob.glob("*")
else:
    lst = glob.glob(sys.argv[1])
if len(lst) == 0: # 빈 리스트
    print("매칭되는 파일이 없습니다")
else:
    for name in lst:
        print(name)
```

실습문제 2

- 문제
  - 모듈 생성 및 사용
  - 파이썬 파일을 하나 생성하고, 이 파일에 간단한 함수를 정의하세요. 그리고 다른 파이썬 파일에서 이 함수를 가져와서 사용하세요.

실습문제 2

- 최종 코드

```
# mymodule.py
def greet(name):
    print(f"Hello, {name}!")


# main.py
import mymodule

mymodule.greet("Alice")
```

실습문제 3

- 문제
  - 모듈에서 여러 함수 사용
  - 모듈을 생성하고 그 안에 두 개의 함수를 정의하세요. 하나는 주어진 숫자의 제곱을 반환하고, 다른 하나는 두 숫자의 합을 반환합니다. 이 함수들을 다른 파일에서 가져와서 사용하세요.

- 요구사항

실습문제 3

- 최종 코드

```
# mathmodule.py
def square(x):
    return x * x

def add(a, b):
    return a + b


# use_mathmodule.py
import mathmodule

print(mathmodule.square(4))  # 16
print(mathmodule.add(5, 3))  # 8
```

실습문제 4

- 문제
  - 모듈의 변수 사용
  - 모듈에 몇 가지 변수를 정의하고, 이를 다른 파일에서 가져와 출력하세요.

실습문제 4

- 최종 코드

```
# config.py
username = "admin"
password = "secret"


# use_config.py
import config

print(config.username)  # admin
print(config.password)  # secret
```

실습문제 5

- 문제
  - 모듈의 클래스 사용
  - 모듈에 클래스를 정의하고, 이 클래스의 인스턴스를 생성하여 다른 파일에서 사용하세요.

실습문제 5

- 최종 코드

```
# personmodule.py
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        return f"Hello, my name is {self.name} and I am {self.age} years
old."

# use_personmodule.py
from personmodule import Person

person = Person("Bob", 30)
print(person.greet())  # Hello, my name is Bob and I am 30 years old.
```

실습문제 6

- 문제
  - **모듈의 예외 처리**
  - 모듈에 사용자 정의 예외를 정의하고, 이 예외를 다른 파일에서 처리하세요.

실습문제 6

- 최종 코드

```
# errormodule.py
class MyError(Exception):
    pass

# handle_error.py
from errormodule import MyError

try:
    raise MyError("Something went wrong")
except MyError as e:
    print(e)
```

실습문제 7

- 문제
  - 패키지 생성 및 모듈 호출
  - my_package라는 패키지를 생성하고, 이 안에 utilities.py 라는 모듈을 만드세요. utilities.py 모듈 안에 multiply_numbers 함수를 정의하고, 다른 파이썬 파일에서 이 함수를 호출하여 결과를 출력하세요.

실습문제 7

- 최종 코드

```
# my_package/utilities.py
def multiply_numbers(a, b):
    return a * b



# main.py
from my_package.utilities import multiply_numbers

result = multiply_numbers(4, 5)
print(f"Result: {result}")  # Result: 20
```

실습문제 8

- 문제
  - 패키지 내 여러 모듈 사용
  - my_package 패키지 안에 math_tools.py와 string_tools.py 두 개의 모듈을 만드세요. math_tools 모듈 안에는 add 함수를, string_tools 모듈 안에는 capitalize_string 함수를 정의하세요. 이 함수들을 메인 스크립트에서 호출하여 결과를 출력하세요.

실습문제 8

- 최종 코드

```
# my_package/math_tools.py
def add(x, y):
    return x + y

# my_package/string_tools.py
def capitalize_string(s):
    return s.capitalize()

# main.py
from my_package.math_tools import add
from my_package.string_tools import capitalize_string

print(add(10, 20))  # 30
print(capitalize_string("hello world"))  # "Hello world"
```