Study Title _____ Date _____

Chapter7. 모듈

7-1 표준 모듈

-모듈은 여러 변수와 함수를 가지고 있는 집합체로 다음과같이 두가지로 분류됩니다.
1. 표준모듈 : 파이썬에 기본적으로 내장되어있는 모듈
2. 외부모듈 : 다른사람들이 만들어서 공개한 모듈

모듈을 가져올 때는 다음과 같은 구문을 사용합니다.
"""
import 모듈 이름
“”

모듈 사용의 기본: math 모듈

-그 이름 그대로 math모듈은 수학과 관련된 기능을 가지고 있습니다.
ex)
math.cos(1)
>>0.543242
math.tan(1)
>>1.557442
math.floor(2.5)
>>2
math.ceil(2.5)
>>3

#from 구문
-모듈에는 정말 많은 변수와 함수가 들어 있습니다. 하지만 그중에서 우리가 활용하고 싶은 기능은 극히 일부 일 수 있으며, math.ceil()처럼 앞에 무언가를 계속 입력하는 것이 귀찮다고 느껴질 수 있습니다.
이때 from구문을 사용합니다.
"""
from 모듈이름 import 원하는 변수 또는 함수
"""
ex)
from math import sin, cos, tan
sin(1)
>>0.8412313
cos(1)
>>0.5413123
tan(1)
>>1.55123121

만약 앞에 math를 붙이는 것이 싫고 모든기능을 가져오는것이 목적이라면 * 기호를 사용합니다.하지만 식별자 이름에서 충돌이 발생할 수 있으니 주의해야합니다.

#as 구문
-모듈의 이름이 너무 길어서 짧게 줄여 사용하고 싶은경우 사용합니다.
“”“
import 모듈 as 사용하고 싶은 식별자
“”“
ex)
import math as m
m.sin(1)
>>0.8413123
m.cos(1)
>>0.5412312

random 모듈

-랜덤한 값을 생성할 때 사용하는 모듈입니다.
ex)
import random

print(random.random())
>>0.56234235
-0부터 1사이의 실수를 리턴해준다

print(random.uniform(10,20))
>>18.21414234
-지정한 범위내의 실수를 리턴해준다

print(random.randrange(10))
>>6
-지정한 범위의 정수를 리턴해준다

print(random.choice([1, 2, 3, 4, 5])
>>2
-리스트 내부에 있는 요소를 랜덤하게 선택해준다

print(random.sample([1, 2, 3, 4, 5], k=2))
>>[5, 4]
-리스트의 요소중에 k개를 선택해준다

#모듈 파일 이름 작성시 주의사항
-모듈은 파일이기때문에 import시 현재폴더에서 모듈을 찾는 것입니다. 따라서 동일한 이름으로 파일을 저장할 경우 기존에 있던 파일과 겹치게 됩니다.

Study Title _____ Date _____

os 모듈

-os모듈은 운영체제와 관련된 기능을 가진 모듈입니다. 새로운 폴더를 만들거나 폴더 내부의 파일목록을 보는 일도 모두 os 모듈을 활용해서 처리합니다.

datetime 모듈

-날짜,시간과 관련된 모듈로, 날짜 형식을 만들 때 자주 사용되는 코드들로 구성되어 있습니다.

ex)

```
import datetime

now = datetime.datetime.now()
print(now.year, "년")
print(now.month, "월")
print(now.day, "일")
print(now.hour, "시")
print(now.minute, "분")
print(now.second, "초")
>>
2019년
4월
23일
3시
51분
41초

output_a = now.strftime("%Y.%m.%d %H:%M:%S")
print(output_a)
>>2019.04.23 03:51:41

output_b = now.strftime("%Y{} %m{} %d{} \
          %H{} %M{} %S{}").format(*"년월일시분초")
print(output_b)
>>2019년 04월 23일 03시 51분 41초

#시간 처리하기
ex)
import date time
now = datetime.datetime.now()

-특정 시간 이후의 시간 구하기
after = now + datetime..timedelta(weeks=1,days=1\
        hours=1, minutes=1, seconds=1)
print(after.strftime("%Y.%m.%d %H:%M:%S"))
>>2019.05.01 03:39:26

-특정 시간 요소 교체하기
output = now.replace(year=(now.year + 1))
print(output.strftime("%Y.%m.%d %H:%M:%S")
>>2020.04.23 02:38:25
```

time 모듈

-time모듈은 유닉스 타임을 구할때, 특정 시간 동안 코드 진행을 정지할때 많이 사용합니다.

ex)

```
import time
print("지금부터 5초 동안 정지합니다.")
time.sleep(5)
print("프로그램을 종료합니다.")
```

urllib 모듈

-이는 URL을 다루는 라이브러리라는 의미입니다. 즉, urllib 모듈은 인터넷 주소를 활용할 때 사용하는 라이브러리입니다.

ex)

```
from urllib import request
target = request.urlopen("https://google.com")
output = target.read()
print(output)
>>구글 웹 페이지에 있는 내용을 읽어서 가져옵니다.
```

7-2 외부 모듈

-파이썬이 기본적으로 제공해 주는 것이 아니라, 다른 사람들이 만들어 배포하는 모듈을 외부 모듈이라 부릅니다.
파이썬으로 인공지능을 개발할 때는 scikit-learn, tensorflow, keras등의 모듈을 활용합니다.

라이브러리와 프레임워크

-라이브러리와 프레임워크를 구분해봅시다.

#라이브러리

-정상적인 제어를 하는 모듈,예를 들어 math모듈은 모듈 내부의 기능을 '개발자'가 직접 호출했습니다. 이처럼 개발자가 모듈의 기능을 호출하는 형태의 모듈을 라이브러리라 합니다.

ex)

```
from math import sin, cos
print(sin(1))
print(cos(1))
```

#프레임워크

-제어 역전이 발생하는 모듈,예를 들어 Flask모듈은 다음과 같이 코드를 작성했습니다.

```
from flask import Flask
app = Flask(__name__

@app.route("/")
def hello():
      return "<h1>Hello World!</h1>"
```

그런데 코드만 보면 내부에 함수만 정의했지 직접적으로 무안가 진행하는 코드는 단 하나도 없습니다.
ex)
set FLASK_APP=flask_basic.py
flask run
*Serving Flask app "flask_basic.py"
*Running on http://127.0.0.1:5000/ (press CTRL+C)

밑에 두줄은 우리가 출력한 적이 없습니다. 그렇다면 이것은 어디서 출력된 것일까요? 바로 Flask모듈 내부입니다. Flask가 내부적으로 서버를 실행한뒤 지정한 파일을 읽어 들여 적절한 상황에 스스로 실행하게 됩니다. 이처럼 모듈이 개발자가 작성한 코드를 실행하는 형태의 모듈을 프레임 워크라고 부릅니다.

개발자가 모듈의 함수를 호출하는 것이 일반적인 제어흐름입니다. 그런데 이와 반대로 개발자가 만든 함수를 모듈이 실행하는 것은 제어가 역전된 것입니다.

7-3 모듈 만들기

-이번 절에서는 원의 반지름과 넓이를 구하는 간단한 모듈을 만들어 봅시다. 먼저 module_basic 디렉터리를 만들어 다음 두파일을 넣어주세요.
main.py test_module.py
main.py가 메인 코드로 활용할 부분입니다.

모듈 만들기

ex)
[module_basic/test_module.py]

```
PI = 3.141592

def number_input():
    output = input("숫자 입력>")
    return float(output)

def get_circumference(radius):
    return 2*PI*radius

def get_circle_area(radius):
    return PI*radius*radius
```

[module_basic/main.py]

```
import test_module as test

radius = test.number_input()
print(test.get_circumference(radius))
print(test.get_circle_area(radius))
```

>>숫자입력> 10
62.83184
314.1592

__name__ ==“__main__”

-다른 사람들이 만든 파이썬 코드들을보다보면 __name__==“__main__”이라는 코드를 많이 볼 수 있습니다. 이 의미가 무엇인지 살펴 보겠습니다.

#__name__
프로그래밍 언어에서는 프로그램의 진입점을 엔트리 포인트 또는 메인이라고 부릅니다. 그리고 이러한 엔트리 포인트 또는 메인내부에서의 __name__은 “__main__”
입니다.

#모듈의 __name__
엔트리 포인트가 아니지만 엔트리 포인트 파일 내에서 import되었기 때문에 모듈 내 코드가 실행 됩니다. 모듈 내부에서 __name__을 출력하면 모듈의 이름을 나타냅니다.
ex)
[module_main/main.py]

```
import text_module

print("메인의 __name__출력하기")
print(__name__)
```

[module_main/test_module.py]

```
print("모듈의 __name__출력하기")
print(__name__)
```

main.py 파일을 실행하면 다음과 같이 출력합니다.
모듈의 __name__ 출력하기
test_module
메인의 __name__ 출력하기
__main__
코드를 실행하면 엔트리 포인트 파일에서는 “__main__”을 출력하지만, 모듈 파일에서는 모듈 이름을 출력하는 것을 볼 수 있습니다.

Study Title _____ Date _____

#__name__활용하기
엔트리 포인트 파일 내부에서는 __name__이 "__main__"이라는 값을 갖기 때문에, 이를 활용하면 현재 파일이 모듈로 실행되는지, 엔트리 포인트로 실행되는지 확인할 수있습니다.
ex)
[module_example/test_module.py]
PI = 3.141592

def number_input():
    output = input("숫자 입력>")
    return float(output)

def get_circumference(radius):
    return 2*PI*radius

def get_circle_area(radius):
    return PI*radius*radius

print("get_circumference(10):",get_circumference(10))
print("get_circle_area(10):", get_circle_area(10))

[module_example/main.py]
import test_module as test

radius = test.number_input()
print(test.get_circumference(radius))
print(test.get_circle_area(radius))

main.py 파일을 실행하면 다음과 같이 출력합니다.
get_circumference(10):62.83184
get_circle_area(10):314.1592
숫자입력>10
62.83184
314.1592

test_module을 모듈로 사용하고 있는데, 내부에서 출력이 발생하니 문제가 됩니다.이를 해결하기위해선, 현재 파일이 엔트리 포인트인지 구분하는 코드를 활용합니다.
예를 들어, test 모듈 내부에

if __name__ == "__main__":
print("get_circumference(10):",get_circumference(10))
print("get_circle_area(10):", get_circle_area(10))

이러한 조건문을 넣어주면 내부에서 출력되는것을 막을 수있습니다.

패키지
-모듈이 모여서 구조를 이루면 패키지라고 부릅니다.

#패키지 만들기
main.py파일은 엔트리 포인트로 사용할 파이썬 파일이며, test_package 폴더는 패키지로 사용할 폴더입니다.
모듈이 모여 구조를 이루면 패키지가 되는것이기 때문에, test_package폴더 내부에 모듈을 하나 이상 넣으면 됩니다. 예로써 module_a.py와 module_b.py 파일을 생성해 보겠습니다.
ex)
[module_package/test_package/module_a.py]
variable_a = "a 모듈의 변수"

[module_package/test_package/module_b.py]
variable_b = "b 모듈의 변수"

[module_package/main.py]
import test_package.module_a as a
import test_package.module_b as b

print(a.variable_a)
print(b.variable_b)

main.py파일을 실행하면 다음과 같이 출력합니다.
a 모듈의 변수
b 모듈의 변수

#__init__.py파일
패키지를 읽을 때 어떤 처리를 수행해야 하거나 패키지 내부의 모듈들을 한꺼번에 가져오고 싶을 때가 있습니다. 이럴 때는 패키지 폴더 내부에 __init__.py 파일을 만들어 사용합니다.
test_package 폴더 내부에__init__.py파일을 만들어 봅시다. 이 파일에서는 __all__이라는 이름의 리스트를 만드는데, 이 리스트에 지정한 모듈들이 전부 읽어 들여집니다.
ex)
[module_package/test_package/__init__.py]
__all__ = ["module_a", "module_b"]
print("test_package를 읽어 들였습니다.")

[module_package/main_1.py]
from test_package import *
print(module_a.variable_a)
print(module_b.variable_b)

main_1.py파일을 실행하면 다음과 같이 출력합니다.
test_package를 읽어 들였습니다.
a 모듈의 변수
b 모듈의 변수