예외처리

김지성 강사

예외처리

✓ 예외처리란?

- ○ 프로그램 처리 중 발생하는 '오류(예외)'를 처리하여 프로그램이 갑자기 중단되지 않고, 예상 가능한 방식으로 동작하도록 돕는 방법
- ○ 발생하는 오류 예시)
  - ■ TypeError: 잘못된 타입 연산
  - ■ FileNotFoundError: 파일을 찾을 수 없을 때
- ○ 예외처리의 **기본 구조**: try-except 구문 (아래의 예제 코드는 모든 예외를 처리하는 코드임)

```
try:
    print(10/0) # 오류가 발생할 가능성이 있는 코드 작성
except:
    print('예외 오류 발생') # 오류가 발생 시 실행할 코드
```

예외처리 - 특정 예외만 처리

✓ 특정 예외만 처리할 경우

- ○ 각 예외를 명시적으로 처리 -> 가독성과 유지보수성이 좋아짐
- ○ 해당예제는 0으로 나눌 때 발생하는 에러인 ZeroDivisionError와 int형으로 변경할 수 없는 값이 들어왔을 때 발생하는 ValueError를 예외처리하는 코드.

```
try:
        x = int(input('숫자를 입력하세요: '))
        print(10/x)
except ZeroDivisionError:
        print('0으로 나눌 수 없습니다.')
except ValueError:
        print('유효한 숫자를 입력하세요.')
```

예외처리 - as 키워드 활용

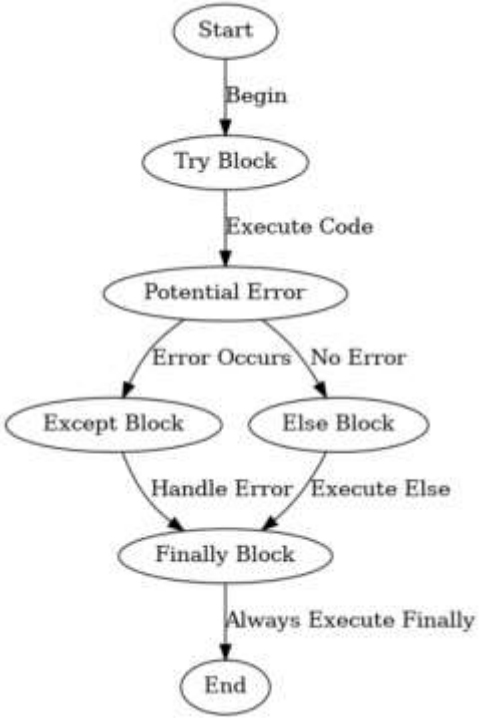
✓ as 키워드 활용하여 별칭을 만들고 그 별칭을 통해 메시지를 출력하는 것을 살펴볼 수 있다.

○ 0을 입력했을 때 발생하는 에러가 객체 e에 담겨진다.

```
try:
        x = int(input('숫자를 입력하세요: '))
        print(10/x)
except ZeroDivisionError as e:
        print(e)  # 출력: division by zero
```

예외처리 - else/finally

✓ except-else-finally 구문을 통해서 조건문과 비슷하게 조건을 컨트롤할 수 있다.




```
try:
        x = int(input('숫자를 입력하세요: '))
        print(10/x)
except ZeroDivisionError:
        print('0으로 나눌 수 없습니다.')
else:
        print('성공적으로 실행되었습니다.')
finally:
        print('프로그램이 종료되었습니다.')
```

예외처리 - else/finally

✓ finally는 예외 발생 여부와 상관없이 항상 실행된다.

- ○ 특정 동작에 대해 반드시 뒤따라 오는 부분, 꼭 해야하는 구문을 작성할 때 try와 finally를 활용하면 좋음
- ○ 파이썬에서는 파일을 읽을 때 open()을 하고 반드시 close()를 통해 객체를 닫아줘야한다. 이럴 때 finally 구문을 활용하면 좋다.

```
try:
    # 파일을 읽는다.
    f = open('file.txt', 'w')

    (... 생략 ...)

finally:
    f.close()  # 중간에 오류가 발생하더라도 무조건 실행된다.
```

예외처리 - 모든 예외 처리

✓ Exception 클래스: 모든 내장 예외의 기본 클래스

- ○ 일반적으로 예외를 구체적으로 처리하는 게 좋지만, 모든 예외를 처리해야 할 경우 사용한다.
- ○ 예외 객체 e를 출력하기(예외 정보 출력) 때문에, 어떤 오류가 발생했는지 정확히 알 수 있음

```
try:
        x = int(input('숫자를 입력하세요: '))
        print(10/x)
except Exception as e:
        print(f'오류가 발생했습니다: {e}')
```

예외처리 연습문제

✓ 사용자로 부터 숫자 2개를 입력받아 split() 메서드를 활용하여 x, y 변수로 unpacking 합니다.

- ○ 만약 사용자가 잘못된 값을 입력하더라도 프로그램이 중단되지 않고 '값을 잘못 입력하셨습니다.' 라는 메시지를 출력할 수 있게 예외처리 부분을 추가해서 완성해보세요.
- ○ 사용자가 제대로된 값을 입력할때까지 입력을 반복합니다.

```
x, y = input('숫자 두 개를 입력하세요.').split()
```

```
숫자 두 개를 입력하세요:  123
값을 잘못 입력하셨습니다. 숫자 두 개를 공백으로 구분하여 다시 입력하세요.
숫자 두 개를 입력하세요:  12 a
값을 잘못 입력하셨습니다. 숫자 두 개를 공백으로 구분하여 다시 입력하세요.
숫자 두 개를 입력하세요:  12 12
입력된 숫자는 x: 12, y: 12입니다.
```

예외처리 연습문제 해답

```
while True:
    try:
        x, y = input("숫자 두 개를 입력하세요: ").split()
        x = int(x)
        y = int(y)
        print(f"입력된 숫자는 x: {x}, y: {y}입니다.")
        break
    except ValueError:
        print("값을 잘못 입력하셨습니다.")
```

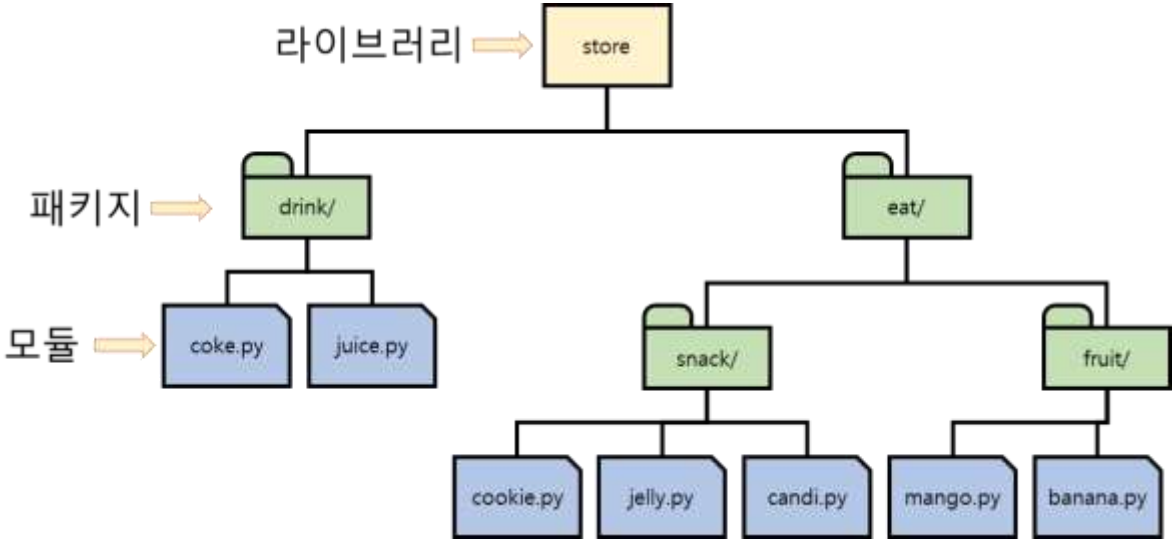
모듈/패키지/라이브러리

김지성 강사

모듈 vs 패키지 vs 라이브러리

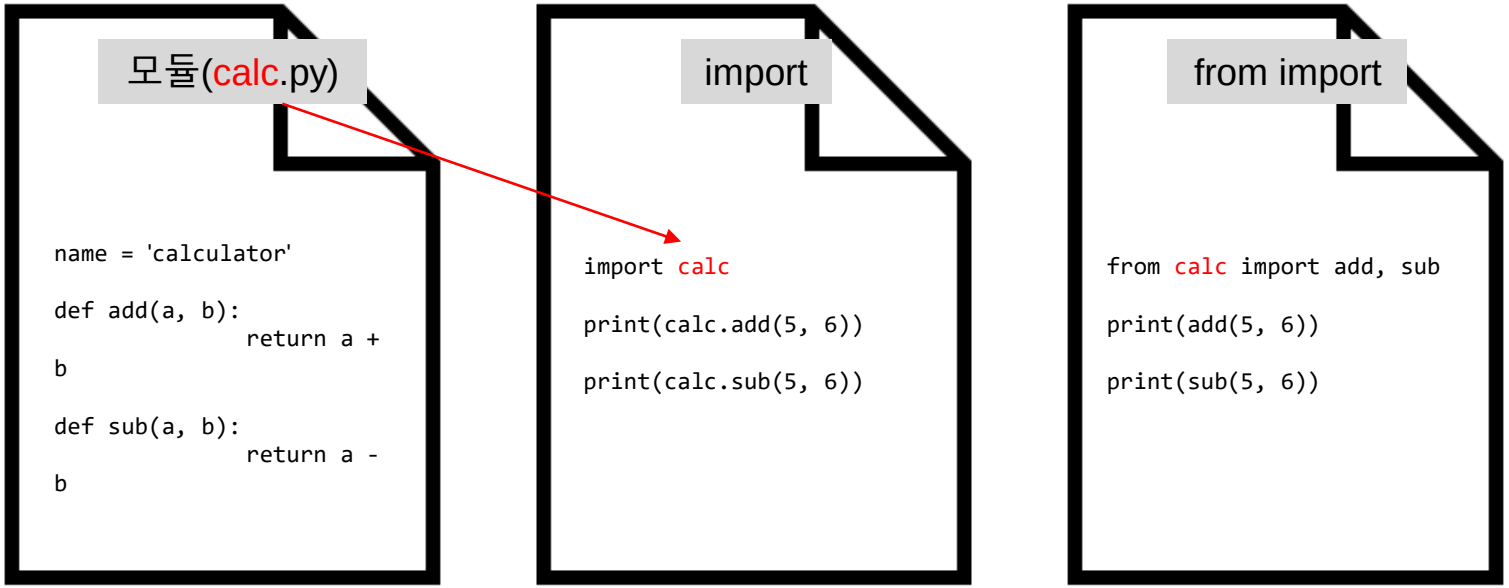
✓ 자주 사용되는 함수와 변수 등을 묶어 모듈을 만들고, 이 모듈들을 모아서 패키지 혹은 라이브러리라고 한다.

- ○ 모듈 : .py 확장자를 갖는 한 개의 파이썬 파일
- ○ 패키지 : 여러 개의 모듈을 폴더에 모아 놓은 것
- ○ 라이브러리 : 패키지를 여러 개를 묶어 설치하여 사용할 수 있도록 제공하는 것



모듈

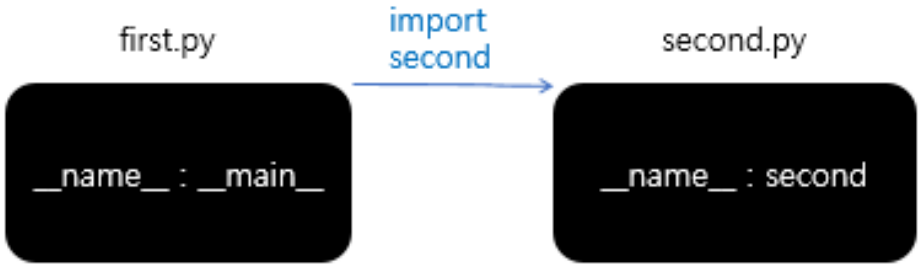
✓ 모듈 : 변수, 함수, 클래스 등을 모아놓은 스크립트 파일



__name__

✓ __name__ (던더네임): 모듈의 이름을 저장해놓은 변수

- ○ **직접 실행된 파일의 name**은 “**main**”이 되고 import 돼서 **모듈로 사용된 파일의 name**은 “**모듈 이름**”이 됨
- ○ python 파일이 직접 실행될 때와 다른 파일에서 임포트되어 사용될때의 동작 구분을 하기 위해 사용됨
- ○ **if __name__ == ‘__main__’:** 구문을 통해 **특정 코드가 테스트용 코드나 메인 스크립트 실행 코드일 때**, 파일이 임포트 되었을 경우 실행되지 않도록 함



__name__

✓ calc.py를 다른 파일에서 실행했을 때의 결과와 직접 calc.py를 실행시켰을 때의 결과를 확인해보자.

모듈(calc.py)

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

def sub(a, b):
                return a - b

my_name = __name__
print('calc.py 파일 : ', my_name)
```

new_file.ipynb

```
import calc
my_name = __name__
print('현재 파일 : ',my_name)
```

실행 결과

```
calc.py 파일 :  calc
현재 파일 :  __main__
```

calc.py 실행

실행 결과

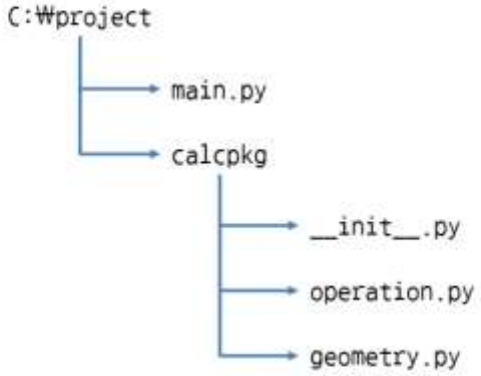
```
[1]:  !python calc.py

      calc.py 파일 :  __main__
```

패키지

✓ 패키지 : 여러 개의 모듈을 폴더에 모아 놓은 것

- ○ __init__.py 파일은 해당 폴더를 패키지로 인식하게 한다.
  - ■ 파이썬 3.3 이상부터는 __init__.py 파일이 없어도 패키지로 인식되지만 호환을 위해 작성
- ○ __init__.py 파일의 내용은 비워둘 수 있다.



패키지

✓ project폴더에 calcpkg 폴더를 만들어서 다음 3개의 파일을 작성.

__init__.py

```
print("__init__.py 파일 실행")
```

geometry.py

```
def triangle_area(base, height):
    return base * height / 2

def rectangle_area(width, height):
    return width * height
```

operation.py

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

def mul(a, b):
    return a * b
```

패키지

✓ 그 후 project 폴더에 main파일을 만들어서 해당 코드를 실행시켜보자.

main.ipynb

```
# calcpkg 패키지의 operation 모듈을 가져옴
import calcpkg.operation

# calcpkg 패키지의 geometry 모듈을 가져옴
import calcpkg.geometry

print('name : ',__name__)

print(calcpkg.operation.add(10, 20))    #
operation 모듈의 add 함수 사용
print(calcpkg.operation.mul(10, 20))    #
operation 모듈의 mul 함수 사용

print(calcpkg.geometry.triangle_area(30, 40))
# geometry 모듈의 triangle_area 함수 사용
print(calcpkg.geometry.rectangle_area(30, 40))
# geometry 모듈의 rectangle_area 함수 사용
```

```
import calcpkg.operation    # calcpkg 패키지의 operation 모듈을 가져옴
import calcpkg.geometry     # calcpkg 패키지의 geometry 모듈을 가져옴

print('name : ',__name__)

print(calcpkg.operation.add(10, 20))    # operation 모듈의 add 함수 사용
print(calcpkg.operation.mul(10, 20))    # operation 모듈의 mul 함수 사용

print(calcpkg.geometry.triangle_area(30, 40))    # geometry 모듈의 triangle_area 함수 사용
print(calcpkg.geometry.rectangle_area(30, 40))   # geometry 모듈의 rectangle_area 함수 사용
```

```
__init__.py 파일 실행
name :  __main__
30
200
600.0
1200
```

모듈과 패키지 경로 찾기

- ✓ site-packages에는 pip으로 설치한 패키지가 들어가있다.
- ✓ 직접 만든 calcpkg는 직접 추가를 해줄 수 있다.

```
import os
import sys

# 현재 작업 디렉토리 가져오기
project_path = os.getcwd()
calcpkg_path = os.path.join(project_path, "calcpkg")

# sys.path에 추가
if calcpkg_path not in sys.path:
    sys.path.append(calcpkg_path)

print(sys.path)
```

```
['C:\\Users\\HOST\\Desktop\\project', 'C:\\Users\\HOST\\anaconda3\\python312.zip', 'C:\\Users\\HOST\\anaconda3\\DLL
s', 'C:\\Users\\HOST\\anaconda3\\Lib', 'C:\\Users\\HOST\\anaconda3', '', 'C:\\Users\\HOST\\anaconda3\\Lib\\site-packa
ges', 'C:\\Users\\HOST\\anaconda3\\Lib\\site-packages\\win32', 'C:\\Users\\HOST\\anaconda3\\Lib\\site-packages\\win32
\\lib', 'C:\\Users\\HOST\\anaconda3\\Lib\\site-packages\\Pythonwin', 'C:\\Users\\HOST\\anaconda3\\Lib\\site-packages
\\setuptools\\_vendor', 'C:\\Users\\HOST\\Desktop\\project\\calcpkg']
```