10. 예외 처리

ㅇ 모든 오류에 대해 잡아내면 프로그램 진행X

작은 오류들은 부드럽게 넘어가자~

예외란

오류

- 구문 오류 : 파이썬 문법에 맞지 않는 문장을 지적하는 오류
  입력 즉시 오류가 나온다.
- 예외 : 문법에 문제는 없지만 실행 중에 부적절한 데이터/상황 때문에 발생하는 오류
  ㄴ 그 뒤는 진행되지 X
- 오류가 나면 프로그램 실행이 중단되고 나머지 부분이 진행되지 않는다.

오류에 대처하는 방법

- 중대한 오류는 해결 후 다시 실행해야 하지만,
- 사소한 오류는 적절한 처리를 하면서 넘어가자. → 오류 지적하고, 프로그램 진행되게!
  → 미리 처리를 하면 프로그램 멈추지 X

예외와 예외 처리

- **예외 발생! 실행 중단!**

```
a = 2
b = '2'
print( a + b )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

에러종류

- **예외 발생! 그러나 부드럽게 넘어감**

```
a = 2
b = '2'
try :
    print( a + b )
except  TypeError :
    print('계산에 문제가 생겼어요.')
```

TypeError가 뜨면, 이것을 띄워줘!

try ~ except

```
try :                              # 아래 문장을 실행하라.
    print( a + b )                 # 이 문장은 살짝 위험해 보인다.
except  TypeError :                # 만일 실행하다가 이 종류의 예외가 발생한다면
  print('계산에 문제가 생겼어요.')          # 이렇게 하라.
```

이 부분의 내용은 꼭 써주는게 better!

언제 사용하나?

- 예외가 발생할 소지가 있는 문장. 그러나 피할 수는 없다.
- 설사 예외가 발생하더라도 부드럽게 넘어갈 수 있다. 계속 실행될 수 있다.
- 예외가 치명적이면 부드럽게 넘어가는 것이 의미가 없다.

예외 처리를 사용할 때에는

- **예외 처리**
  - 예외가 발생할 소지가 있는 문장만
  - 어떤 예외가 발생할지
  - 예외 처리의 내용으로 무엇을 넣을지
  - 프로그램의 나머지 부분을 계속 진행할 수 있을지
- **예외의 내용을 사용자에게 알리려면** (두 예시 모두 이렇게 사용된다하고 넘어가기)

```
try
    print( a + b )
except TypeError as err :
    print ( format(err))
```

(e 라는 에러가 발생하면 형식에 맞춰 에러를 표시한다! (시스템에 부합하도 그대로))

```
import  traceback
try
    print( a + b )
except TypeError :
    traceback.print_exec()
```

(조금 더 보기쉽게!)

else, finally

- 예외가 발생하면 except, 발생하지 않으면 else, 항상 실행하려면 finally

```
① try :
      f = open(filename, 'r')
② except IOError:
      print('파일을 찾지 못했습니다')
③ else :
      print('파일을 찾았습니다')
  # 파일 읽기 작업 진행
      f.close()
④ finally :
      print(filename, '처리 완료')
```

파일이 없으면 이런 에러가 남

예외발생 : ① → ② → ④ 순

예외없음 : ① → ③ → ④ 순

예외가 발생했든 안했든 해야할것

실습1

- 파일 이름을 입력 받아 그 내용을 화면에 출력하는 프로그램을 작성하라. 단, 파일이 존재하지 않으면 다시 이름을 입력받아라.