CHAPTER 8. 예외 처리

section121. 예외 처리의 기본

step 1.
알고가기

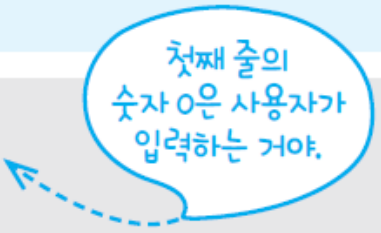
- 핵심내용
  - 예외 처리란 프로그램을 정지시킬 수 있는 오류를 예방하기 위한 처리이다.

- 코딩하기 전에
  - 예외란 프로그램 실행 중 프로그램이 갑자기 정지될 수 있는 사건을 말한다.
  - try 블록에는 오류가 발생할 수 있는 코드를 넣는다.
  - except 블록에는 오류가 발생했을 때 처리할 코드를 넣어준다. 반드시 1개 이상의 except 블록이 필요하다.
  - else 블록은 except 블록이 실행되지 않을 경우 실행되는 코드를 넣는다.

section121. 예외 처리의 기본

step 2.
코딩

```
1:  # section_121.py
2:
3:  number = int(input('나눌 숫자를 입력하세요: '))
4:
5:  try:
6:      result = 10/number
7:  except ZeroDivisionError:
8:      print('0으로 나누면 안돼요.')
9:  else:
10:     print(result)
```




```
나눌 숫자를 입력하세요: 0
0으로 나누면 안되요.
```

section121. 예외 처리의 기본

step 3.
코드분석

- try:

  result = 10/number
  - 예외가 발생할 수 있는 코드를 try 블록에 작성한다.
  - 이 코드는 0으로 나눌 경우 예외가 발생할 수 있다.
- except ZeroDivisionError:

  print('0으로 나누면 안돼요.')
  - 만약 사용자가 프로그램 실행 중 0을 입력하면 파이썬 인터프리터는 ZeroDivisionError이라는 예외를 발생시킨다.
  - 따라서 프로그래머는 ZeroDivisionError예외를 대비해 except 블록을 만들어줘야 한다.

section121. 예외 처리의 기본

step 3.
코드분석

- else:
  print(result)
  - try 블록에서 아무런 문제가 없어서 무난히 넘어갔을 때 실행된다.
  - 계산의 결과값을 출력해주는 코드이다.
- 개발자는 프로그램이 작동하는 도중 발생할 수 있는 예외를 최대한 처리해 주는게 좋다.
- 프로그래머는 파이썬이 제공하는 모든 예외 목록을 알아야 한다.

section121. 예외 처리의 기본

step 4.

3줄 요약

• 3줄 요약 •

- 예외란 프로그램 실행 중 발생할 수 있는 오류를 말한다.
- 예외 처리란 프로그램 실행 중에 발생한 오류 때문에 프로그램이 정지되는 것을 막기 위한 방법을 말한다.
- try 블록에 오류가 생길 수 있는 코드를 넣고, 해당 오류를 처리하기 위한 except 문을 구성한다.

section122. 예상되는 오류가 여러 개일 때

step 1.
알고가기

- 핵심내용
  - 예외가 여러 개 예상될 때는 해당 오류에 대한 except문을 추가한다.

section122. 예상되는 오류가 여러 개일 때

step 2.
코딩

```
1:  # section_122.py
2:
3:  try:
4:      number = int(input('나눌 숫자를 입력하세요 '))
5:      result = 10/number
6:  except ValueError:
7:      print('숫자만 입력하세요.')
8:  except ZeroDivisionError:
9:      print('0으로 나누면 안돼요.')
10: except:
11:     print('2가지 외에 개발자가 전혀 예측하지 못한 에러ㅠㅠ')
12: else:
13:     print(result)
```

```
나눌 숫자를 입력하세요: 0
0으로 나누면 안되요.
```

section122. 예상되는 오류가 여러 개일 때

step 3.
코드분석

- except ValueError:
  - 기본적으로 예측가능한 예외는 개발자가 최대한 except 문으로 처리해준다.
- except:
  - 예외 종류를 지정하지 않으면 앞서 지정되지 않은 모든 예외 상황을 받아준다.
- 오류명을 생략하는 것이 편할 수 있지만, 개발자가 정확한 오류를 예측할 수 없으면 사용자에게는 적절한 피드백을 줄 수 없고, 개발자 스스로는 적절한 처리를 할 수 없기 때문에 좋은 것만은 아니다.

section122. 예상되는 오류가 여러 개일 때

step 4.

3줄 요약

• 3줄 요약 •

- 예외가 발생할 수 있는 종류가 여러 개라면 종류별로 except문을 만든다.
- except문에 오류명을 생략하면 모든 종류의 오류를 받을 수 있다.
- except문 옆에 두 개 이상의 오류를 나열할 수 있다.

section123. as 문

step 1.
알고가기

- 핵심내용
  - as문을 이용하면 예외 정보를 얻을 수 있다.

- 코딩하기 전에
  - as문을 이용하면 파이썬이 제공해주는 오류 정보를 활용할 수 있다.
  - 형식: except <오류 종류> as <변수명>

section123. as 문

step 2.
코딩

```
1:  # section_123.py
2:
3:  def openFile():
4:      file = open('없는파일.txt')
5:      line = file.readline()
6:      number = int(line.strip())
7:
8:  try:
9:      openFile()
10: except OSError as err:
11:     print('시스템 에러: ', err)
12: except:
13:     print('내가 예측할 수 없는 오류ㅠ')
14: else:
15:     print(number)
```

```
시스템 에러: [Errno 2] No such file or directory: '없는파일.txt'
```

section123. as 문

step 3.
코드분석

- except OSError as err:
  - 파일 입출력 오류를 담당하는 OSError가 발생할 경우 이 줄이 실행된다.
  - OSError에 대한 정보를 err이라는 변수에 저장한다.
  - err은 err.args라고 쓸 수 있다. 원래 err.args인데, 클래스 내부의 __str__() 메서드에 같은 내용이 정의되어 있어서 클래스 인스턴스의 이름만 적어도 동일한 내용이 출력된다.
  - err.args는 튜플의 형태로 정보를 반환하기 때문에 err.args[0]처럼 인덱스를 이용해서 필요한 정보만 가져올 수도 있다.

section123. as 문

step 4.

3줄 요약

• 3줄 요약 •

- as문을 이용해 파이썬이 제공해 주는 예외에 대한 정보를 얻을 수 있다.
- as 〈변수명〉 일 때, '변수' 대신 '변수.args'를 이용할 수도 있다.
- args는 튜플의 형태기 때문에 인덱싱으로 일부분만 가져올 수 있다.

section124. finally문

step 1.
알고가기

- 핵심내용
  - try문이 실행된 이후 무조건 실행해야 하는 코드는 finally문에 넣는다.

- 코딩하기 전에
  - finally 문은 예외가 발생해도 실행되고, 예외가 없어도 실행된다.
  - 즉, finally문은 try 문이 실행된 이후 무조건 실행된다
  - 보통 try문에서 시작한 작업을 마무리하기 위한 코드를 넣는다.
  - finally 문을 넣었을 때는 except를 생략할 수 있다.

section124. finally문

step 2.
코딩

```
1:  # section_124.py
2:
3:  def writeFile():
4:      try:
5:          f = open('myfile', 'w')
6:          try:
7:              f.write('Hello World!')
8:          finally:
9:              f.close()
10:     except IOError:
11:         print('oops!')
12:
13: def readFile():
14:     try:
15:         f = open('myfile', 'r')
16:         line = f.readline()
17:     except IOError:
18:         print('oops!')
19:     else:
20:         print(line)
21:     finally:
22:         f.close()
23:
24: writeFile()
25: readFile()
```

section124. finally문

step 3.
코드분석

- 예외처리의 전체 구조

```
try :
    〈오류가 발생할 수 있는 코드〉
except 〈오류 종류1〉:
    〈오류 종류1이 발생했을 때 처리할 코드〉
except (〈오류 종류2〉, 〈오류 종류3〉, 〈오류 종류4〉):
    〈오류 종류2, 3, 또는 4가 발생했을 때 처리할 코드〉
except 〈오류 종류5〉 as 변수명:
    〈오류 종류5가 발생했을 때 처리할 코드〉
else:
    〈오류가 발행하지 않았을 때 처리할 코드〉
finally:
    〈예외 발생과 무관하게 무조건 실행할 코드〉
```

section124. finally문

step 4.

3줄 요약

• 3줄 요약 •

- finally문은 try문이 실행된 이후 무조건 실행된다.
- 따라서 예외의 발생과 무관하게 무조건 실행해야 하는 코드를 넣는데 적합하다.
- finally문이 있을 땐 except문을 생략할 수 있다.

section125. raise문

step 1.
알고가기

- 핵심내용
  - raise문은 개발자가 의도적으로 예외를 발생시킬 때 사용한다.

- 코딩하기 전에
  - raise문은 프로그램을 개발하는 과정에서 try...except 문이 잘 작동하는지 확인하는 차원에서 필요할 수도 있고, 다른 개발자에게 어떤 작업을 강제하도록 하기 위해서 사용할 수도 있다.

section125. raise문

step 2.
코딩

```
1:  # section_125.py
2:
3:  try:
4:      raise IndexError
5:  except IndexError:
6:      print('인덱스 에러 발생')
```

```
인덱스 에러 발생
```

section125. raise문

step 3.
코드분석

- raise IndexError
  - 개발자가 강제로 IndexError를 발생시킨다.
- except IndexError:
  - IndexError 에러가 발생했으니 당연히 except 중 IndexError 에러를 처리하는 부분을 찾고 블록을 실행한다.

section125. raise문

step 4.

3줄 요약

• 3줄 요약 •

- raise문은 개발자가 의도적으로 예외를 발생시킬 때 사용한다.
- raise문을 이용하면서 에러 정보를 개발자가 작성할 수도 있다.
- 개발자가 작성한 에러 정보는 as문을 이용하여 얻을 수 있다.

section126. 사용자 정의 예외

step 1.
알고가기

- 핵심내용
  - 파이썬은 사용자가 직접 예외를 만들어 쓸 수 있는 방법을 제공한다.

- 코딩하기 전에
  - 파이썬이 제공하는 내장 예외에 원하는 예외 목록이 없다면 개발자가 스스로 사용자 예외를 만들 수 있다.

section126. 사용자 정의 예외

step 2.
코딩

```
1:  # section_126.py
2:
3:  import math
4:
5:  class QuadError(Exception):
6:      pass
7:
8:  def quad(a,b,c):
9:      if a == 0:
10:         qe = QuadError("이차 방정식이 아니에요.")
11:         qe.member= (a, b, c)
12:         raise qe
13:     if b*b-4*a*c < 0:
14:         qe = QuadError("방정식의 근이 없어요.")
15:         qe.member= (a, b, c)
16:         raise qe
17:     x1 = (-b+math.sqrt(b*b-4*a*c))/(2*a)
18:     x2 = (-b-math.sqrt(b*b-4*a*c))/(2*a)
19:     return (x1, x2)
20:
21: def getQuad( a, b, c ):
22:     try:
23:        x1, x2 = quad( a, b, c )
24:        print("방정식의 근은", x1, x2)
25:     except QuadError as err:
26:        print(err, err.member)
27:
28: getQuad(0, 100, 10)
```

잠깐! 이차 방정식을 모르는 사람을 위해서...
이차 방정식의 형태가 $ax^2 + bx + c = 0$일 때, 근의 공식은
$x = -b \pm \frac{\sqrt{b^2-4ac}}{2a}$ 야

section126. 사용자 정의 예외

step 3.
코드분석

- class QuadError(Exception):
  - 모든 사용자 정의 예외는 Exception 클래스나 그 하위 클래스를 상속받아서 구현한다.
  - QuadError 클래스는 Exception 클래스를 상속받았고, 클래스에는 아무 내용이 없다.
- qe = QuadError("이차 방정식이 아니에요.")
  - 객체변수 qe에 QuadError() 클래스를 생성해서 할당했다.
  - 이때 "이차 방정식이 아니에요."라는 에러에 대한 정보도 함께 보냈다.
- qe.member= (a, b, c)
  - qe 객체에 member 변수를 추가하고 여기에 튜플 (a, b, c)를 저장했다.
  - a, b, c는 사용자가 입력한 2차방정식의 계수이다.

section126. 사용자 정의 예외

step 3.
코드분석

- raise qe
  - 사용자 정의 예외 QuadError를 강제로 발생시킨다.
  - 왜냐하면 a==0일 경우 이차방정식이 아니기 때문이다. 그러나 파이썬은 이런 예외를 위한 어떠한 예외도 제공하지 않는다. 즉, 개발자의 몫이다.

section126. 사용자 정의 예외

step 4.

3줄 요약

• 3줄 요약 •

- 파이썬은 내장 예외 외에 사용자 정의 예외도 제공한다.
- 사용자 정의 예외는 사용자가 예외를 직접 만들어서 사용하는 것이다.
- 사용자 정의 예외는 Exception 또는 그 하위 클래스를 상속받아야 한다.