파이썬 프로그래밍
강의 노트 #10

예외 처리

오류

- 프로그래밍의 오류
  - 문법 오류
  - 논리 오류
  - 실행 오류
- 문법 오류
  - 프로그래밍 언어를 잘못 사용해서 발생
  - 파이썬이 어디서 오류가 발생했는지 알려줌

오류

```
print("오류 발생 확인 프로그램")
n = int(input("정수를 입력하세요: ")
print(f"사용자가 입력한 정수: {n}")
```

코드10-2

```
[실행 결과]
File "SyntaxError.py", line 2
 n = int(input("정수를 입력하세요: ")
         ^
SyntaxError: '(' was never closed
```

오류

- 논리 오류
  - 문제 해결 방법을 잘못 지정한 것
- 실행 오류
  - 문법적/논리적으로 문제가 없는데 오류가 발생하는 것
  - 예
    - 사용자가 정수로 변환할 수 있는 문자열을 입력해야 하는데 잘못 입력
    - 파일을 여는데 이미 삭제되었거나 존재하지 않는 파일일 수 있음
    - 파일에서 데이터를 읽거나 쓸 때, 하드디스크에 오류가 있거나 파일에 문제가 있어 동작하지 않을 수 있음

오류

- 사용자가 정수로 변환할 수 있는 문자열을 잘못 입력한 예

```
>>> n = int(input("정수를 입력하세요: "))
정수를 입력하세요: 23.5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with
base 10: '23.5'
```

- 실행 오류가 발생하는 몇 가지 예
  - int() 명령에 실수 형태로 된 문자열을 전달
  - 문자열에서 index() 함수를 사용하는데 찾는 문자열 없음
  - open() 함수를 이용해서 파일을 열 때 파일이 없는 경우
  - 리스트에 있는 요소 개수보다 큰 범위의 요소를 접근
  - 0으로 나누는 경우

오류 대응 코드의 필요성

- 지금까지 우리는 프로그램을 만들면서 항상 오류가 발생하지 않는다고 가정했음
- 하지만, 프로그램을 사용하다 보면 예기치 못했던 오류가 발생하는 경우가 많음
- 오류를 모두 처리하는 것은 어렵지만, 프로그램 실행이 멈추는 것을 최소화
- 프로그램을 개발할 때 문제가 발생할 수 있는 부분을 생각하고 적절하게 오류에 대응하는 코드를 작성하는 방어적 코딩을 해야 함
- 파이썬에서는 예외처리 기능을 이용해서 다양한 오류 상황을 처리할 수 있도록 지원

예외처리 기법

- 예외(exception)은 프로그램을 실행하면서 발생할 수 있는 오류를 통칭
- 파이썬에서는 try: … except: 구문을 이용해서 예외처리를 지원
- 아래는 기본적인 사용 방법을 보여줌

```
try:
    오류를 발생시킬 수 있는 파이썬 코드
except [오류 종류 [as 오류 변수]]:
    오류가 발생했을 때 실행시킬 코드
[except [오류 종류 [as 오류 변수]]:
    또 다른 종류의 오류가 발생했을 때 실행시킬 코드
…   # 또 다른 except 구문
```

예외처리 기법

- []에 있는 내용은 생략 가능
- try구문은 최소한 한 개 이상의 except 구문과 연동되어야 함
- 코드 10-2에 예외처리 추가

```
print("예외 처리 확인 프로그램")
try:
    s = input("정수를 입력하세요: ")
    n = int(s)                                  # (1)
    print(f"사용자가 입력한 정수: {n}")          # (2)
except:
    print(f"정수로 변환할 수 없습니다. 입력값: {s}") # (3)
print("프로그램 종료")                           # (4)
```

예외처리 기법

- 예외가 발생했을 때의 실행 순서

```
print("예외 처리 확인 프로그램")
try:
    s = input("정수를 입력하세요: ")
    n = int(s)                                                    # (1)
    print(f"사용자가 입력한 정수: {n}")                              # (2)
except:
    print(f"정수로 변환할 수 없는 값을 입력했습니다. 입력값: {s}") # (3)
print("프로그램 종료")                                            # (4)
```

① 23.5

② 예외 발생

③

④

- 예외가 발생하지 않을 때는?

여러 가지 오류가 발생하는 코드

- 여러 가지 오류가 발생할 수 있음

```
lst = [1, 2, 3]
idx = int(input("변경할 요소의 위치(인덱스)를
입력하세요: "))
n = int(input("새로운 값을 입력하세요: "))     # (1)
lst[idx] = n # idx 위치의 요소 값을 n으로 변경 # (2)
print(lst)
```

  - (1)에서 정수가 아닌 문자열이 입력될 때 ValueError
  - (2)에서 idx가 3 이상일 때 IndexError

여러 오류에 대응하는 예외 처리 코드 작성

- 여러 가지 오류가 발생할 수 있는 경우, 예외 처리 방법
  - 모든 오류에 대해서 한꺼번에 대응
  - 오류마다 각각 대응
  - 여러 가지 오류 중 일부만 대응하고, 나머지는 무시하거나 한꺼번에 대응

여러 오류에 대응하는 예외 처리 코드 작성

- 모든 오류를 함께 처리
  - except를 오류 정보 없이 사용

```
try:
    lst = [1, 2, 3]
    idx = int(input("변경할 요소의 위치(인덱스)를 입력하세요: "))
    n = int(input("새로운 값을 입력하세요: "))
    lst[idx] = n # idx 위치의 요소 값을 n으로 변경
    print(lst)
except: # 모든 오류를 한 곳에서 대응
    print("오류: 인덱스 범위를 벗어났거나 정수가 아닌 문자열을 입력했음")
```

여러 오류에 대응하는 예외 처리 코드 작성

- 각 오류를 따로 처리
  - except에 각 오류 정보를 명시

```
try:
    lst = [1, 2, 3]
    idx = int(input("변경할 요소의 위치(인덱스)를
입력하세요: "))
    n = int(input("새로운 값을 입력하세요: "))
    lst[idx] = n # idx 위치의 요소 값을 n으로 변경
    print(lst)
except ValueError: # ValueError에 대응
    print("오류: 정수가 아닌 문자열을 입력했음")
except IndexError: # IndexError에 대응
    print("오류: 인덱스 범위를 벗어남")
```

여러 오류에 대응하는 예외 처리 코드 작성

- 일부 오류만 대응하고 나머지는 무시

```
try:
    lst = [1, 2, 3]
    idx = int(input("변경할 요소의 위치(인덱스)를
입력하세요: "))
    n = int(input("새로운 값을 입력하세요: "))
    lst[idx] = n # idx 위치의 요소 값을 n으로 변경
    print(lst)
except ValueError: # ValueError에 대응
    print("오류: 정수가 아닌 문자열을 입력했음")
```

여러 오류에 대응하는 예외 처리 코드 작성

- 일부 오류만 대응하고 나머지는 한꺼번에 처리

```
try:
    lst = [1, 2, 3]
    idx = int(input("변경할 요소의 위치(인덱스)를
입력하세요: "))
    n = int(input("새로운 값을 입력하세요: "))
    lst[idx] = n # idx 위치의 요소 값을 n으로 변경
    print(lst)
except ValueError: # ValueError에 대응
    print("오류: 정수가 아닌 문자열을 입력했음")
except: # 나머지 (IndexError, NameError)에 대응
    print("오류 발생")
```

오류의 예

오류 종류	설명
ValueError	인자로 다른 자료형의 값을 전달
IndexError	자료구조에서 인덱스의 범위를 벗어남
NameError	값이 저장되지 않은 변수가 사용
UnicodeDecodeError	파일이 저장된 인코딩 방식과 지정된 인코딩 방식이 다름
TypeError	허용되지 않은 자료형 사용
FileNotFoundError	파일이 없음
ZeroDivisionError	0으로 나눔

UnicodeDecodeError

- data.utf8.txt파일이 utf-8 방식으로 한글 문자열을 저장했다고 가정할 때, cp949 형식으로 열고 읽으면 UnicodeDecodeError 발생

```
>>> f = open("data.utf8.txt")
>>> f.read()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'cp949' codec can't decode byte
0xed in position 0: illegal
multibyte sequence
```

FileNotFoundError

- 파일 작업할 때 파일이 없으면 발생

```
filename = "C:\\temp\\noname.txt"
f = open(filename)
f.close()
```

- C:₩temp₩noname.txt가 없다고 가정할 때 실행 결과

```
Traceback (most recent call last):
File "exception8.py", line 2, in <module>
f = open(filename)
FileNotFoundError: [Errno 2] No such file or
directory: 'C:\\temp\\noname.txt'
```

스크립트 실행 중에 종료하기

- sys모듈의 exit함수를 이용하면 스크립트 실행 중에 종료하기

```
import sys
try:
    fname = "c:\\temp\\a.ini"
    f = open(fname)
    lines = f.readlines()
    f.close()
except FileNotFoundError: # FileNotFoundError 발생
    print("Could not open " + fname)
    sys.exit()
except:   # 다른 오류 일괄 처리
    print("Other error occurred")
print("End of the program")
```

실습문제 1

- 문제
  - 사용자로부터 파일 이름을 입력받고 내용을 화면에 출력하는 프로그램 작성
- 요구사항
  - 사용자가 입력한 파일이 존재하지 않으면 한 번 더 입력할 수 있도록 함
  - 두 번째 입력한 파일은 있다고 가정
  - 파일이 없는지 확인하는 것은 예외처리를 사용

실습문제 1

- 최종 코드

```
filename = input("파일 이름을 입력하세요: ")
try:
    f = open(filename)
except FileNotFoundError:
    filename = input("파일이 없습니다. 다시 파일
이름을 입력하세요: ")
    f = open(filename)
for line in f:
    print(line.strip())
f.close()
```

실습문제 2

- 문제
  - 사용자가 입력한 파일 이름을 가지고 파일의 내용을 화면에 출력하는 프로그램 작성
  - 파일은 cp949 또는 utf-8 인코딩 방식으로 저장되었지만, 어떤 방식인지는 알 수 없음
- 요구사항
  - 파일이 어떤 인코딩 방식으로 저장되었는지 모름
  - 예외 처리를 이용해서 인코딩 방식을 찾아낼 것

실습문제 2

- 최종 코드

```
filename = input("파일 이름 입력: ")
try:
    f = open(filename, encoding = "cp949")
    s = f.read()
except UnicodeDecodeError:
    f = open(filename, encoding = "utf-8")
    s = f.read()
f.close()
# s에는 파일의 내용이 있음
print(s)
```

예외 처리로 pass문 사용

- 실행 오류가 발생했을 때, 오류를 처리하지 않고 프로그램 실행을 지속시키기 위해 pass문을 사용하는 경우가 있음

```
try:
    print(3 / 0)   # 오류 발생
except ZeroDivisionError:
    pass           # 예외 처리로 아무것도 안함
print("오류가 발생함에도 여기까지 출력됨")
```