05

파이썬 날개 달기

05-4 예외 처리

05-5 내장 함수

05-4 예외 처리

▪ 오류는 어떤 때 발생하는가?

```
>>> f = open("나없는파일", 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: '나없는파일'
```

```
>>> 4 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> a = [1,2,3]
>>> a[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

05-4 예외 처리

▪ 예외 처리 기법

- **try, except문**
  - 오류 처리를 위한 구문
  - try 블록 수행 중 오류가 발생하면 except 블록 수행
    try 블록에서 오류가 발생하지 않으면 except 블록 미수행

```
try:
    ...
except [발생 오류[as 오류 메시지 변수]]:
    ...
```

  - except 구문

```
except [발생 오류 [as 오류 메시지 변수]]:
```

    - [ ] 기호
      - 괄호 안의 내용을 생략할 수 있다는 관례 표기 기법

05-4 예외 처리

▪ 예외 처리 기법

▪ try, except문

- except 구문 사용법

1) try, except만 쓰는 방법

- 오류 종류에 상관없이
  오류가 발생하면 except 블록 수행

```
try:
    …
except:
    …
```

2) 발생 오류만 포함하는 방법

- 오류가 발생했을 때 except문에
  미리 정해 놓은 오류 이름과 일치할 때만
  except 블록을 수행한다는 뜻

```
try:
    …
except 발생 오류:
    …
```

05-4 예외 처리

▪ 예외 처리 기법

▪ try, except문

- except 구문 사용법
  - 3) 발생 오류와 오류 메시지 변수를 포함한 except문
    - 오류가 발생했을 때 except문에 미리 정해 놓은 오류 이름과 일치할 때만 except 블록을 수행하고, 오류 메시지의 내용까지 알고 싶을 때 사용하는 방법

```
try:
    ...
except 발생 오류 as 오류 메시지 변수:
    ...
```

```
try:
    4 / 0
except ZeroDivisionError as e:
    print(e)
```

```
결괏값: division by zero
```

05-4 예외 처리

▪ 예외 처리 기법

- **try ... finally**
  - finally절은 try문 수행 도중 예외 발생 여부에 상관없이 항상 수행됨
  - 보통 finally절은 사용한 리소스를 close해야 할 때에 많이 사용함
    - 예) foo.txt 파일을 쓰기 모드로 열어 try문을 수행한 후
      예외 발생 여부와 상관없이 finally절에서 f.close()로 열린 파일을 닫을 수 있음

```
f = open('foo.txt', 'w')
try:
    # 무언가를 수행한다.
finally:
    f.close()
```

05-4 예외 처리

예외 처리 기법

여러 개의 오류 처리하기

- try문 안에서
  여러 개의 오류를 처리하기 위한 방법

```
try:
    ...
except 발생 오류 1:
    ...
except 발생 오류 2:
    ...
```

- 예) 0으로 나누는 오류와 인덱싱 오류 처리

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError:
    print("0으로 나눌 수 없습니다.")
except IndexError:
    print("인덱싱할 수 없습니다.")
```

05-4 예외 처리

▪ 예외 처리 기법

▪ 여러 개의 오류 처리하기

- 오류 메시지 가져오기

```
try:
    a = [1,2]
    print(a[3])
    4/0
except ZeroDivisionError as e:
    print(e)
except IndexError as e:
    print(e)
```

- 2개 이상의 오류를 동시에 처리하기 위해 괄호를 사용하여 함께 묶어 처리

```
try:
    a = [1,2]
    print(a[3])
    4/0
except (ZeroDivisionError, IndexError) as e:
    print(e)
```

05-4 예외 처리

▪ 오류 회피하기

- 특정 오류가 발생할 경우 그냥 통과시키는 방법

```
try:
    f = open("나없는파일", 'r')
except FileNotFoundError:  ◀── 파일이 없더라도 오류를 발생시키지 않고 통과한다.
    pass
```

- try문 안에서 FileNotFoundError가 발생할 경우에 pass를 사용하여 오류를 그냥 회피하도록 함

05-4 예외 처리

▪ 오류 일부러 발생시키기

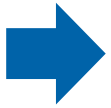
- raise 명령어를 사용해 오류를 강제로 발생시킬 수 있음
  - 예) Bird 클래스를 상속받는 자식 클래스가 반드시 fly라는 함수를 구현하도록 하고 싶은 경우
    - 파이썬 내장 오류 NotImplementedError와 raise문 활용
    - fly 함수를 구현하지 않은 상태로 fly 함수 호출 시 NotImplementedError 오류 발생

```
class Bird:
    def fly(self):
        raise NotImplementedError
```

```
class Eagle(Bird):
    pass

eagle = Eagle()
eagle.fly()
```

← Eagle 클래스는 Bird 클래스를 상속 받음



```
Traceback (most recent call last):
  File "...", line 33, in <module>
    eagle.fly()
  File "...", line 26, in fly
    raise NotImplementedError
NotImplementedError
```

05-4 예외 처리

▪ 예외 만들기

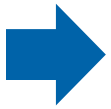
- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능

```
class MyError(Exception):
    pass
```

- 예) 별명을 출력해주는 함수에서 MyError 사용하기

```
def say_nick(nick):
    if nick == '바보':
        raise MyError()
    print(nick)
```

```
say_nick("바보")
```



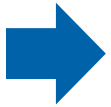
```
Traceback (most recent call last):
  File "...", line 11, in <module>
    say_nick("바보")
  File "...", line 7, in say_nick
    raise MyError()
__main__.MyError
```

05-4 예외 처리

▪ 예외 만들기

- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능
  - 예) 예외처리 기법을 사용하여 MyError 발생 예외 처리

```
try:
    say_nick("천사")
    say_nick("바보")
except MyError:
    print("허용되지 않는 별명입니다.")
```



```
천사
허용되지 않는 별명입니다.
```

05-4 예외 처리

▪ 예외 만들기

- 파이썬 내장 클래스인 Exception 클래스를 상속하여 생성 가능
  - 예) MyError에서 __str__ 메서드 구현하여 오류 메시지 사용하기

```
class MyError(Exception):
    def __str__(self):
        return "허용되지 않는 별명입니다."
```

```
try:
    say_nick("천사")
    say_nick("바보")
except MyError as e:
    print(e)
```

```
천사
허용되지 않는 별명입니다.
```

05-5 내장 함수

파이썬 내장 함수

- 외부 모듈과 달리 import 등 기타 설정 없이 바로 사용 가능

abs(x)

- x의 절댓값을 돌려주는 함수

```
>>> abs(3)
3
```

```
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

all(x)

- 반복 가능한(Iterable) 자료형 x가 모두 참이면 True, 하나라도 거짓이면 False 반환

```
>>> all([1, 2, 3])
True
```

```
>>> all([1, 2, 3, 0])
False
```

any(x)

- x가 모두 거짓이면 False, 하나라도 참이면 True 반환

```
>>> any([1, 2, 3, 0])
True
```

```
>>> any([0, ""])
False
```

05-5 내장 함수

chr(x)

- 아스키(ASCII) 코드를 입력받아 코드에 해당하는 문자 반환

```
>>> chr(97)
'a' ◀── 아스키 코드 97은 소문자 a
>>> chr(48)
'0' ◀── 아스키 코드 48은 숫자 0
```

dir(x)

- 객체가 자체적으로 가지고 있는 변수나 함수 반환

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

divmod(a, b)

- a를 b로 나눈 몫과 나머지를 튜플 형태로 반환

```
>>> divmod(7, 3)
(2, 1) ◀── 7 나누기 3의 몫은 2, 나머지는 1
```

05-5 내장 함수

enumerate(x)

- '열거하다'라는 뜻
- 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아 인덱스 값을 포함하는 enumerate 객체 반환

```
>>> for i, name in enumerate(['body', 'foo', 'bar']):
...     print(i, name)
...
0 body
1 foo
2 bar
```

eval(expression)

- 실행 가능한 문자열(expression)을 입력으로 받아 문자열을 실행한 결괏값 반환
- 문자열로 파이썬 함수나 클래스를 동적으로 실행할 때 사용

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

05-5 내장 함수

filter(f, iterable)

- Iterable 자료형의 요소가 함수 f에 입력되었을 때 반환 값이 참인 것만 묶어서 반환

```
#filter1.py
def positive(x):
    return x > 0

print(list(filter(positive, [1, -3, 2, 0, -5, 6])))
```

```
결괏값: [1, 2, 6]
```

- lambda도 사용 가능

```
>>> list(filter(lambda x: x > 0, [1, -3, 2, 0, -5, 6]))
```

hex(x)

- 정수 값을 입력받아 16진수로 변환

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

05-5 내장 함수

id(object)

- object(객체)의 고유 주소 값 반환

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

3, a, b는 모두 같은 객체를 가리킴

input([prompt])

- 사용자 입력을 받는 함수
- 문자열 인자 생략 가능
- 매개변수로 문자열을 주면 프롬프트 띄움

```
>>> a = input()
hi
>>> a
'hi'
>>> b = input("Enter: ")
Enter: hi
```

사용자가 입력한 정보를 변수 a에 저장

Enter: 프롬프트를 띄우고 사용자 입력을 받음

05-5 내장 함수

int(x)

- 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 변환

```
>>> int('3') ◀── 문자열 형태 '3'
3
>>> int(3.4) ◀── 소수점이 있는 숫자 3.4
3
```

- int(x, radix)
  - radix 진수 문자열을 10진수로 변환

```
>>> int('1A', 16)
26
```

instance(object, class)

- object: 인스턴스 / class: 클래스 이름
- 인스턴스가 클래스의 인스턴스인지 판단하여 참이면 True, 거짓이면 False 반환

```
>>> class Person: pass ◀── 아무 기능이 없는 Person 클래스 생성
...
>>> a = Person() ◀── Person 클래스의 인스턴스 a 생성
>>> isinstance(a, Person) ◀── a가 Person 클래스의 인스턴스인지 확인
True
>>> b = 3
>>> isinstance(b, Person) ◀── b가 Person 클래스의 인스턴스인지 확인
False
```

05-5 내장 함수

len(x)

- 입력값의 길이(요소의 전체 개수) 반환

```
>>> len("python")
6
```

```
>>> len([1,2,3])
3
```

list(iterable)

- Iterable 자료형을 리스트로 변환

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

map(f, iterable)

- Iterable 자료형의 각 요소를 함수 f가 수행한 결과를 묶어서 반환

```
>>> def two_times(x): return x*2
...
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

- lambda 활용 가능

```
>>> list(map(lambda a: a*2, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

05-5 내장 함수

max(iterable)

- Iterable 자료형의 최댓값 반환

```
>>> max([1, 2, 3])
3
```

```
>>> max("python")
'y'
```

min(iterable)

- Iterable 자료형의 최솟값 반환

```
>>> min([1, 2, 3])
1
```

```
>>> min("python")
'h'
```

oct(x)

- 정수 형태의 숫자를 8진수 문자열로 변환

```
>>> oct(34)
'0o42'
>>> oct(12345)
'0o30071'
```

05-5 내장 함수

open(filename, [mode])

- filename: 파일 이름 / mode: 읽기 방법
- mode를 생략하면 기본값인
  읽기 전용 모드(r)로 파일 객체 반환
- b는 w, r, a와 함께 사용

모드	설명
w	쓰기 모드로 파일 열기
r	읽기 모드로 파일 열기
a	추가 모드로 파일 열기
b	바이너리 모드로 파일 열기

```
>>> f = open("binary_file", "rb")
```

ord(c)

- 문자의 아스키 코드 값 반환

```
>>> ord('a')
97
```

```
>>> ord('0')
48
```

pow(x, y)

- x의 y 제곱한 결괏값 반환

```
>>> pow(2, 4)
16 ◀── 2의 4 제곱
```

```
>>> pow(3, 3)
27 ◀── 3의 3 제곱
```

05-5 내장 함수

- **range([start,] stop [,step])**
  - 입력받은 숫자에 해당하는 범위 값을 Iterable 객체로 반환

1) 인수가 하나일 경우
   - 시작 숫자를 지정해 주지 않으면 range 함수는 0부터 시작

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

2) 인수가 2개일 경우
   - 시작 숫자와 끝 숫자
   - 끝 숫자는 해당 범위에 포함되지 않음

```
>>> list(range(5, 10))
[5, 6, 7, 8, 9] ◀── 끝 숫자 10은 포함되지 않음
```

3) 인수가 3개일 경우
   - 세 번째 인수는 숫자 사이의 거리

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9] ◀── 1부터 9까지, 숫자 사이의 거리는 2
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9] ◀── 0부터 -9까지, 숫자 사이의 거리는 -1
```

05-5 내장 함수

round(number[, ndigits])

- 숫자를 입력받아 반올림해 주는 함수

```
>>> round(4.6)
5
>>> round(4.2)
4
```

- ndigits는 반올림하여 표시하고 싶은 소수점의 자릿수

```
>>> round(5.678, 2)
5.68
```

sorted(iterable)

- 입력값을 정렬한 후 그 결과를 리스트로 반환

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

05-5 내장 함수

str(object)

- 객체를 문자열 형태로 변환

```
>>> str(3)
'3'
```

```
>>> str('hi'.upper())
'HI'
```

sum(iterable)

- 리스트나 튜플의 모든 요소의 합 반환

```
>>> sum([1,2,3])
6
```

```
>>> sum((4,5,6))
15
```

tuple(iterable)

- Iterable 자료형을 튜플 형태로 변환

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```

05-5 내장 함수

type(object)

- 입력값의 자료형 반환

```
>>> type("abc")
<class 'str'> ◀── "abc"는 문자열 자료형
>>> type([])
<class 'list'> ◀── [ ]는 리스트 자료형
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'> ◀── 파일 자료형
```

zip(*iterable)

- 동일한 개수로 이루어진 자료형을 묶어주는 역할을 하는 함수

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

감사합니다

“Life is too short,

You need Python!”

인생은 너무 짧으니,

파이썬이 필요해!