python

#11

TODAY

- 예외 처리 (try-except)
- IF문과 논리 연산

>>> 예외 처리

예외 처리

- 지금까지 python을 같이 배워오면서 Error 메시지를 경험하지 않은 사람은 아마 없을 것
- 그렇다면 이러한 현상이 반드시 지양되어야 하는가?

예외 처리

- 특히 사용자의 입력을 받는 코드를 작성한다면 입력 실수에 의한 예외가 발생할 가능성이 높다.
- 코드 작성 시 예외를 최소화하는 것도 있지만, 예외가 발생했을 때 처리하는 것도 중요

try-except

```
# 기본적으로는 if-else와 유사함

text = '200%'

try:
    number = int(text)
except ValueError:
    print('{}는 숫자가 아니잖아요'.format(text))
```

try-except

```
def safe_pop_print(list, index):
    try:
        print(list.pop(index))
    except IndexError:
        print('Index:{}인 값 없음'.format(Index)
safe_pop_print([1, 2, 3], 5)
```

if-else로 바꾸면?

```
def safe_pop_print(list, index):
    if index < len(list):
        print(list.pop(index))
    else:
        print('Index:{}인 값 없음'.format(Index)
safe_pop_print([1, 2, 3], 5)
```

모든 오류를 한 번에?

```
try:
    list = []
    print(list[0]) # IndexError
    text = 'abc'
    number = int(text) # invalid literal
except:
    print('오류가 발생했습니다.')
```

모든 오류를 한 번에?

```
try:
    list = []
    print(list[0]) # IndexError
    text = 'abc'
    number = int(text) # invalid literal
except Exception as ex:
    print('오류가 발생했습니다.', ex)
```

예외를 직접 일으키기

```
def rsp(mine, yours):
    allowed = ['가위', '바위', '보']
    if mine not in allowed:
        raise ValueError
    if yours not in allowed:
        raise ValueError
rsp('가위', '바')
```

예외를 직접 일으키기

```
try:
    rsp('가위', '바')
except ValueError:
    print('잘못된 값을 넣었습니다')
```

예외를 직접 일으키기

```
def rsp(mine, yours):
    allowed = ['가위', '바위', '보']
    if mine not in allowed:
        raise ValueError('가위바위보 값이 아닙니다')
    if yours not in allowed:
        raise ValueError('가위바위보 값이 아닙니다')
rsp('가위', '바')
```

중첩된 반복문에서 활용

```
classrooms = {'1반': [162, 175, 198, 137, 145, 199], '2반': [165, 177, 157, 160, 191]}

for class_id, heights in classrooms.items():
    for height in heights:
        if height > 190:
            print(class_id, '에 190이 넘는 학생이 있습니다')
            break
```

중첩된 반복문에서 활용

```
classrooms = {'1반': [162, 175, 198, 137, 145, 199], '2반': [165, 177, 157, 160, 191]}

for class_id, heights in classrooms.items():
    for height in heights:
        if height > 190:
            print(class_id, '에 190이 넘는 학생이 있습니다')
            raise StopIteration
```

>>> if문과 논리 연산

if-elif-else

- 첫 조건문에 해당될 때...
- 첫 조건문에 해당되지 않으면서 이번 조건문에 해당될 때...
- 어떤 조건문에도 해당되지 않을 때...

logic.py

```
a = 10

if a < 0 and 2 ** a > 1000 and a % 5 == 2 and round(a) == a:

    print('복잡한 식')
```

AND 연산

전부 AND 연산으로 이루어져 있기 때문에, 맨 앞의 조건이 틀리면 뒷부분은 볼 것도 없다.

OR 연산

전부 OR 연산으로 이루어져 있기 때문에, 맨 앞의 조건이 맞다면 뒷부분은 볼 것도 없다.

단락 평가

AND나 OR 연산을 할 때 첫번째 값을 보고 더 이상 실행할 필요가 없으면 두번째 이상의 값은 실행되지 않는다.

단락 평가의 활용

두번째 조건이 첫번째 조건을 전제하고 있다고 해도 if문에 함께 넣어줄 수 있다.

단락 평가가 이루어지지 않는다면 두번째 조건에서 오류가 생김

단락 평가의 활용

```
dictionary = {'K2': 'V2'}

if 'K1' in dictionary and dictionary['K1']='V1':
    print('맞네')
else:
    print('아니네')
```

bool 값과 논리 연산

- bool 값: True or False
- 정수/실수의 경우 0이면 False, 0이 아니면 True

bool 값과 논리 연산

- 리스트의 경우 빈 리스트는 False, 나머지는 True
- 문자열의 경우 빈 문자열(None)은 False, 나머지는 True

bool_test.py

```
value = input('입력>') or '입력값 없음'

print('입력받은 값:', value)
```

THANK YOU