Study Title _____ Date _____

Chapter6. 예외처리

6-1 구문 오류와 예외

오류의 종류

-프로그래밍 언어의 오류에는 크게 두가지 종류가 있습니다.
syntax error : 실행 전에 발생하는 오류
exception or runtime error : 실행 중에 발생하는 오류

#Syntax error
ex)
```
print("#프로그램이 시작되었습니다.")
print("#예외를 강제로 발생시켜볼께요!)
>>SyntaxError ; EOL while scanning string literal
```

코드를 실행하면 EOL(End of Line)에 문제가 있다고 합니다.구문에 오류가 있어 프로그램이 실행조차 되지않는 오류입니다.이런경우는 그냥 닫는 땅모표로 문자열을 닫아서 해결할 수 있습니다.

#Exception or Runtime Error
ex)
```
print("# program is started!")
list_a[1]
>>NameError : name 'list_a' is not defined
```

코드를 실행하면 첫번째 문자열이 출력되고 그다음 에러가 발생합니다.이처럼 실행중에 발생하는 오류를 excption or runtime error라 합니다.

기본 예외 처리

-예외를 해결하는 모든것을 예외 처리라고 부릅니다.
예외를 처리한는 방법
1.조건문을 사용하는 방법
2.try구문을 사용하는방법
먼저 조건문을 사용해서 예외를 처리하는 방법부터 살펴보겠습니다.

#예외 상황 확인하기
ex)
```
number_input_a = int(input("정수입력>"))
print("반지름:", number_input_a)
print("원의둘레:", 2*3.14*number_input_a)
pirnt("원의넓이:", 3.14*number_input_a**2)
```

만약 정수를 입력하면 정상적으로 실행하지만 문자를 입력할경우 ValueError가 발생합니다.

#조건문으로 예외 처리하기
-예외처리를 정수를 입력하지 않았을때를 조건으로 해보겠습니다.
isdigit()함수를 사용해 숫자로만 구성된 글자인지 확인하면 예외를 피할수 있습니다. 다음 코드를 봅시다.
Ex)
```
user_input_a = input("정수입력>")
if user_input_a.isdigit():
    number_input_a = int(user_input_a)
    print("반지름:", number_input_a)
    print("원의둘레:", 2*3.14*number_input_a)
    pirnt("원의넓이:", 3.14*number_input_a**2)
else:
    print("정수를 입력하지 않았습니다.")
```

이러면 프로그램이 중간에 강제로 죽지않고 정상으로 종료됩니다.

try except 구문

-프로그래밍 언어의 구조적인 문제로인해 조건문만으로 예외 처리할 수 없는 경우도 있습니다. 그래서 요즘 프로그래밍 언어는 예외를 처리할 수 있는 구문을 제공합니다.
```
"""
try:
    예외가 발생할 가능성이 있는 코드
except:
    예외가 발생했을 때 실행할 코드
"""
```
이전의 예제를 try except구문으로 변경해 보겠습니다.
ex)
```
try:
    number_input_a = int(input("정수입력>"))
    print("반지름:", number_input_a)
    print("원의둘레:", 2*3.14*number_input_a)
    pirnt("원의넓이:", 3.14*number_input_a**2)
except:
    print("무언가 잘못되었습니다.")
```
코드를 실행하고 문자열을 입력하면 프로그램이 강제로 종료되는 일 없이 예외 처리를 하고 정상적으로 종료됩니다.

#try except구문과 pass키워드 조합하기
-예외가 발생하면 일단 처리해야 하지만, 해당 코드가 딱히 중요한 부분이 아니라면, 그냥 pass키워드를 넣고 넘어갑니다.
```
"""
try:
    예외가 발생할 가능성이 있는 코드
except:
    pass
""
```

Study Title _____ Date _____

try except else 구문

-이 구문은 예외가 발생할 가능성이 있는 코드만 try 구문 내부에 넣고 나머지를 모두 else 구문으로 빼는 경우가 많습니다.

```
"""
try:
    예외가 발생할 가능성이 있는 코드
except:
    예외가 발생했을때 실행할 코드
else:
    예외가 발생하지 않았을 때 실행할 코드
"""
```

ex)

```
try:
    number_input_a = int(input("정수입력>"))
except:
    print("정수를 입력하지않았습니다.")
else:
    print("반지름:", number_input_a)
    print("원의둘레:", 2*3.14*number_input_a)
    pirnt("원의넓이:", 3.14*number_input_a**2)
```

다른프로그래밍 언어에는 else구문이 없습니다.
한마디로 꼭 이렇게 코드를 작성할 필요는없습니다.
else구문을 사용하지않고 try구문 내부에 모두 넣고 처리해도 상관없습니다.

finally 구문

-이 구문은 예외 처리 구문에서 가장 마지막에 사용할 수 있는 구문입니다.예외가 발생하든 발생하지 않든 무조건 실행할때 사용하는 코드입니다.

```
"""
try:
    예외가 발생할 가능성이 있는 코드
except:
    예외가 발생했을때 실행할 코드
else:
    예외가 발생하지 않았을 때 실행할 코드
finally:
    무조건 실행할 코드
```

#try,except,finally 구문의 조합

-예외 처리 구문은 다음과 같은 조합들이있습니다.

1. try + except
2. try + except + else
3. try + except + finally
4. try + finally

이 외의 조합은 실행했을때 구문 오류가 발생합니다.

#finally에 대한 오해

-일반적으로 finally 키워드를 설명하는 예제로 파일처리를 많이 사용합니다. 하지만, 실제 finally 구문을 사용하는 것과는 전혀 관련이 없습니다. finally키워드는 어떤 조건에 무조건 사용해야 하는게 아니라 finally키워드를 사용하면 코드가 깔끔해질 것 같다고 생각되는 경우에 사용하빈다. 어떤 경우에 코드가 깔끔해질 수 있는지 살펴봅시다.

#try구문 내부에서 return키워드를 사용하는 경우

-finally구문은 반복문 또는 함수내부에 있을때 위력을 발휘합니다.

ex)

```
def test():
    pirnt("test()함수의 첫 줄입니다.")
    try:
        print("try구문이 실행되었습니다.")
    except:
        print("except구문이 실행되었습니다.")
    else:
        print("else구문이 실행되었습니다.")
    finally:
        print("finally 구문이 실행되었습니다.")
    print("test()함수의 마지막 줄입니다.")
test()
>>
test()함수의 첫 줄입니다.
try 구문이 실행되었습니다.
finally 구문이 실행되었습니다.
```

try구문 중간에서 탈출해도 finally구문은 무조건 실행됩니다.따라서 코드를 깔끔하게 만들고 싶을때 finally구문을 활용하는 경우가 많습니다.

#반복문과 함께 사용하는 경우

-반복문에서 break로 빠져나갈 때도 마찬가지입니다.

ex)

```
print("프로그램이 시작되었습니다.")
while True:
    try:
        print("try구문이 실행되었습니다.")
        break
        print("try구문의 break키워드 뒤입니다.")
    except:
        print("except구문이 실행되었습니다.")
    finally:
        print("finally구문이 실행되었습니다.")
    print("while 반복문의 마지막 줄입니다.)
>>
프로그램이 시작되었습니다.
try구문이 실행되었습니다.
finally구문이 실행되었습니다.
```

Study Title _____ Date _____

6-2 예외 고급

-프로그래밍언어에서 예외가 발생하면 예외와 관련된 정보가 생깁니다. 그리고 이러한 예외 정보는 예외 객체에 저장됩니다.

"""

try:
    예외가 발생할 가능성이 있는 구문

except 예외의 종류 as 예외 객체를 활용할 변수 이름:
    예외가 발생했을 때 실행할 구문

“”

예외 객체

-예외의 종류에는 모든 예외를 통들어 뜻하는 Exception을 사용합니다.

ex)

```
try:
    number_input_a = int(input(“정수입력>”))
    print(“반지름:”, number_input_a)
    print(“원의둘레:”, 2*3.14*number_input_a)
    pirnt(“원의넓이:”, 3.14*number_input_a**2)
except Exception as exception:
    print(type(exception))
    print(exception)
>>
정수입력> yes
<class ‘ValueError’>
exception: invalid literal for int() with base 10: ‘yes’
```

예외 구분하기

-예외 객체를 사용하면 except구문을 if조건문처럼 사용해서 예외를 구분할 수 있습니다.

“”

```
try:
    예외가 발생할 가능성이 있는 구문
except 예외의 종류A:
    예외A가 발생했을 때 실행할 구문
except 예외의 종류B:
    예외B가 발생했을 때 실행할 구문
—반복—
```

ex)

```
list_number = [52, 273, 32, 72, 100]
try:
    number_input = int(input(“정수입력>”))
    print(“{}번째 요소: {}”.format(number_input,\
     list_number[number_input]))
except ValueError:
    print(“정수를 입력해 주세요!”)
except IndexError:
    print(“리스트의 인덱스를 벗어났어요!”)
>>
정수입력> yes
정수를 입력해 주세요
>>
정수입력> 100
리스트의 인덱스를 벗어났어요!
```

모든 예외 잡기

-except구문으로 예외를 구분했을때, 만약 예외조건에 일치하는 것이 없다면 당연히 예외가 발생하며 프로그램이 강제 종료됩니다.

ex)

```
list_number = [52, 273, 32, 72, 100]
try:
    number_input = int(input(“정수입력>”))
    print(“{}번째 요소: {}”.format(number_input,\
     list_number[number_input]))
    예외.발생해주세요()
except ValueError as exception:
    print(“정수를 입력해주세요!”)
    print(type(exception), exception)
except IndexError as exception:
    print(“리스트의 인덱스를 벗어났어요!”)
    print(type(exception), exception)
>>
정수입력>1
1번째 요소: 273
NameError : name ‘예외’ is not defined
```

이렇게 예외가 발생해 강제로 종료됩니다. 따라서 else 구문처럼 마지막에는 모든예외를 뜻하는 Exception을 넣어서 프로그램이 죽지 않게 하는 것이 좋습니다.

raise 구문

-개발하는동안에 아직 구현하지 않은 부분이기때문에 일부로 강제종료시키는 경우도 있습니다.

"""

raise 예외 객체

“”

ex)

```
number = input(“정수입력>”)
number = int(number)
if number > 0:
    raise NotImplementedError
else:
    raise NotImplementedError
```