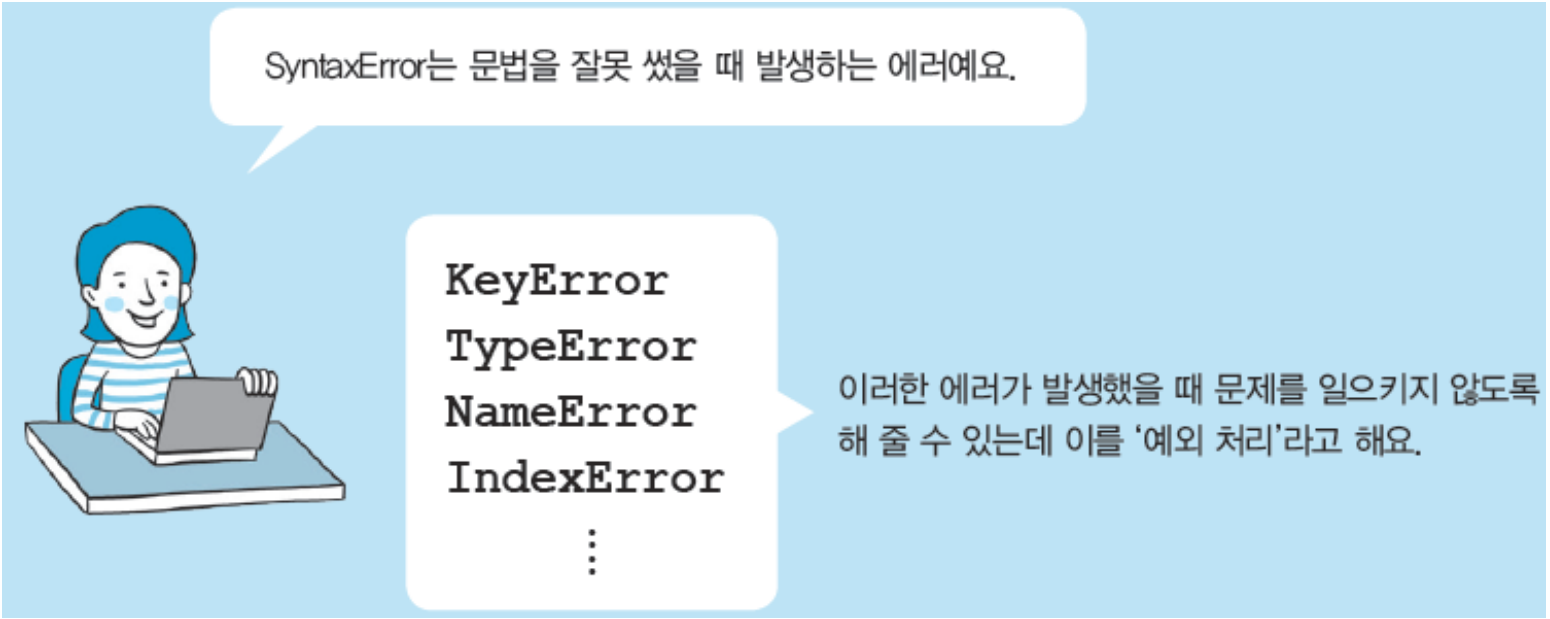
16. 에러와 예외처리

1. 에러와 예외
2. 에러 종류
3. 예외 처리 기초
4. else 블록 이해하기
5. finally 블록 이해하기
6. 예외를 직접 발생시키기 – raise
7. 정리

1. 에러와 예외

- 문법 에러 - Syntax error
- 그외 에러 - TypeError, NameError, IndexError, ValueError, KeyError



1. 에러와 예외

◆ 에러와 예외

- 파이썬에서 발생하는 모든 에러들을 묶어서 '예외'라고 함.
- SyntaxError 예제 (문법이 틀린 경우)

>>> print('hello) SyntaxError: EOL while scanning string literal	print( ) 함수의 인수에 따옴표가 빠졌어요.
>>> print('hello')) SyntaxError: invalid syntax	괄호가 잘못 되었어요.

>>> a = 10
>>> if a == 10
SyntaxError: invalid syntax
	if 줄 끝에 콜론(:)이 빠졌어요.

>>> a = 10; b = 20
>>> a + = b
SyntaxError: invalid syntax
	+= 기호는 붙여서 써야 합니다.

>>> a =< b SyntaxError: invalid syntax	=< 기호는 없습니다. <=이라고 써야죠.

1. 에러와 예외

- NameError, ZeroDivisionError, IndexError

①	②	③
❶ a = 10 ❷ b = 20 ❸ c = 30 ❹ print(a) ❺ print(b) ❻ print(d) ← NameError ❼ print(c)	❶ m = 10 ❷ n = 15 ❸ p = m / n ← 분모가 15임. ❹ print(p) ❺ n = 0 ❻ q = m / n ← ZeroDivisionError ❼ print(p) ← 분모가 0임.	❶ print('hello') ❷ A = [10,20,30] ❸ print(A[0]) ❹ print(A[3]) ← indexError ❺ print(A[1]) ❻ print('good bye')

①	②	③
10 20 Traceback ..... NameError: name 'd' is not defined	0.6666666666666666 Traceback ..... ZeroDivisionError: division by zero	hello 10 Traceback ..... IndexError: list index out of range

1. 에러와 예외

◆ KeyError

```
>>> info = {'name':'Alice', 'address':'Seoul', 'age':20, 'cell_phone':'010-111-0000'}
>>> info['age']                    # 사전에서는 키가 인덱스입니다.
20
>>> info['mail_address']           # 없는 키를 넣으면 KeyError가 발생합니다.
……
KeyError: 'mail_address'
```

1. 에러와 예외

◆ EAFP 코딩 스타일

- 에러가 발생하지 않게 하려면, if 구문을 적절히 이용해서 코드를 작성할 수 있음. 그런데, 파이썬에서는 'EAFP' 코딩 스타일을 권장함.
- EAFP 코딩 스타일

> **E**asier to **A**sk for **F**orgiveness than **P**ermission. This common Python coding style assumes the existence of valid keys or attributes and catches exceptions if the assumption proves false. This clean and fast style is characterized by the presence of many try and except statements. The technique contrasts with the LBYL style common to many other languages such as C.

- 예외 처리를 권장함.

2. 에러 종류

- AttributeError
  - 모듈에 있는 속성을 잘못 사용한 경우

```
>>> import math
>>> print(math.pii)    # math 모듈에는 pi 속성이 있습니다.
... ...
AttributeError: module 'math' has no attribute 'pii'
>>> math.abs(-10)    # math 모듈에는 abs( ) 함수가 없습니다. (abs( )는 내장 함수로 존재)
... ...
AttributeError: module 'math' has no attribute 'abs'
```

2. 에러 종류

- AttributeError
  - 클래스에 있는 속성을 잘못 사용한 경우

```
〉〉〉 L = [1,3,5]
〉〉〉 L.append(7)        # append( )는 리스트의 메소드입니다.
〉〉〉 L.add(10)          # add( )는 리스트에 없는 메소드입니다.
… …
AttributeError: 'list' object has no attribute 'add'
〉〉〉 dir(list)          # 리스트에 있는 메소드를 확인해 보세요. ‘add’는 없어요.
[…… 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
'reverse', 'sort']
```

2. 에러 종류

- IndexError

```
>>> L = [1,3,5]
>>> print(L[5])     # 5는 없는 인덱스
.....
IndexError: list index out of range
```

```
>>> name = 'Alice'
>>> print(name[10])    # 10은 없는 인덱스
.....
IndexError: string index out of range
```

```
>>> L = [5,7]
>>> L.pop()    # 마지막 원소 삭제
7
>>> L.pop()
5
>>> L.pop()    # 빈 리스트
.....
IndexError: pop from empty list
```

```
>>> L = [1, 3, 5, 7, 9]
>>> L.pop(1)    # 인덱스 1의 원소 삭제, 반환
3
>>> L.pop(5)     # 인덱스 5는 없음
.....
IndexError: pop index out of range
```

2. 에러 종류

- KeyError – 집합과 사전에서 발생함.
  - 빈 집합에서 pop() 메소드 적용하거나, 집합에 없는 원소를 remove()할 때, KeyError 발생

```
>>> S = {10, 15}
>>> S.pop()
10
>>> S.pop()
15
>>> S.pop()          # S는 빈 집합
.....
KeyError: 'pop from an empty set'
```

```
>>> S = {1,2,3}
>>> S.remove(2)
>>> S.remove(5)          # 5는 없는 원소
.....
KeyError: 5
```

2. 에러 종류

- KeyError – 집합과 사전에서 발생함.
  - 사전에 없는 키를 이용하면, KeyError 발생

```
>>> D = {1:'one', 2:'two'}
>>> print(D[3])          # 3은 D에 없는 키입니다.
......
KeyError: 3
```

```
>>> D = {1:'one', 2:'two', 3:'three'}
>>> D.pop(7)      # 7은 없는 키
......
KeyError: 7
```

```
>>> D = {'name':'Alice', 'age':10}
>>> D.popitem()
('age', 10)
>>> D.popitem()
('name', 'Alice')
>>> D.popitem()      # D가 빈 사전이에요.
......
KeyError: 'popitem(): dictionary is empty'
```

2. 에러 종류

- ValueError - 값을 가져올 수 없는 경우에 발생.
  - 리스트에 없는 원소 x에 대해서 remove(x)하거나, index(x) 속성을 적용할 때 발생함.

```
>>> L = [1,3,5]
>>> L.remove(3)      # 3은 있는 데이터
>>> L.remove(10)     # 10은 없는 데이터
.....
ValueError: list.remove(x): x not in list
```

```
>>> L = [10, 20]
>>> L.index(20)     # 20은 있는 데이터
1
>>> L.index(30)     # 30은 없는 데이터
.....
ValueError: 30 is not in list
```

2. 에러 종류

- ValueError - 값을 가져올 수 없는 경우에 발생.
  - 리스트에 없는 원소 x에 대해서 remove(x)하거나, index(x) 속성을 적용할 때 발생함.

```
>>> L = [1,3,5]
>>> L.remove(3)      # 3은 있는 데이터
>>> L.remove(10)     # 10은 없는 데이터
.....
ValueError: list.remove(x): x not in list
```

```
>>> L = [10, 20]
>>> L.index(20)     # 20은 있는 데이터
1
>>> L.index(30)     # 30은 없는 데이터
.....
ValueError: 30 is not in list
```

  - 튜플에서도 없는 원소 x에 대해서 index(x) 적용하면 에러 발생함.

```
>>> T = (5,6,7)
>>> T.index(6)       # 6은 있는 데이터
1
>>> T.index(10)      # 10은 없는 데이터
.....
ValueError: tuple.index(x): x not in tuple
```

2. 에러 종류

- FileNotFoundError - 없는 파일을 open()할 때 발생함.

```
>>> f = open('data.txt')    # 만약에 'data.txt'가 없는 데이터이면 에러가 발생합니다.
.....
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

2. 에러 종류

◆ TypeError

- 자료형에 맞는 연산을 수행하지 않는 경우 TypeError 발생함.
- 연산시에 고려해야할 부분들

- 수치 자료형인 정수(int), 실수(float), 복소수(complex)끼리는 연산이 가능합니다.
- 수치 자료형과 부울 자료형도 섞어서 연산이 가능합니다.
- 시퀀스 자료형(문자열, 리스트, 튜플)은 같은 자료형끼리만 + 연산이 가능합니다.
- 집합과 사전에는 '+' 연산을 할 수가 없습니다.

2. 에러 종류

◆ TypeError

〈 리스트 + 튜플 〉

```
〉〉〉 L = [1,2,3] ; T = (4,5)
〉〉〉 L + T              # list + tuple
......
TypeError: can only concatenate list (not "tuple") to list
```

➡ 해결

```
〉〉〉 L + list(T)
[1, 2, 3, 4, 5]
〉〉〉 tuple(L) + T
(1, 2, 3, 4, 5)
```

〈 리스트 + 문자열 〉

```
〉〉〉 L = ['melon', 'pear']
〉〉〉 string = 'apple'
〉〉〉 L + string              # list + str
......
TypeError: can only concatenate list (not "str") to list
〉〉〉 L + list(string)        # 문자열에 list( ) 함수를 적용하면 다음과 같은 결과예요.
['melon', 'pear', 'a', 'p', 'p', 'l', 'e']
```

2. 에러 종류

◆ TypeError

〈 문자열 + 정수 〉	
⟩⟩⟩ name = 'alice'; age = 10 ⟩⟩⟩ name + age # str + int ..... TypeError: must be str, not int	➡ 해결 ⟩⟩⟩ name + str(age) 'alice10' ▶ NOTE int(name) + age 는 안 됩니다.

2. 에러 종류

◆ TypeError

〈 리스트 + 정수 〉

```
〉〉〉 L = [10, 20, 30]
〉〉〉 L + 40              # list + int
......
TypeError: can only concatenate list (not "int") to list
〉〉〉 L + list(40)
......
TypeError: 'int' object is not iterable # list 함수 인수는 Iterable 자료형이어야 함.
〉〉〉 int(L) + 40
......
TypeError: int( ) argument must be a string, a bytes-like object or a number, not 'list'
```

➡ 해결

```
〉〉〉 L.append(40)     # append 메소드로만 int를 리스트와 연결할 수 있어요.
〉〉〉 L
[10, 20, 30, 40]
```

2. 에러 종류

◆ TypeError - '*' 연산은 '시퀀스 * 정수'만 가능함.

```
>>> L = [1, 2, 3]
>>> L * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> L * 3.0      # float는 안 됩니다.
.....
TypeError: can't multiply sequence by non-int of type 'float'
```

➡ 해결

```
>>> L * int(float('3.0'))
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

L * int('3.0') 은 안 됩니다.

```
>>> string = 'hello'
>>> string * 3
'hellohellohello'
>>> string * '3.0'      # float 는 안 됩니다.
.....
TypeError: can't multiply sequence by non-int of type 'str'
```

➡ 해결

```
>>> string * int(float('3.0'))
'hellohellohello'
```

string * int('3.0') 은 안 됩니다.

2. 에러 종류

- TypeError - 함수의 인수에 잘못된 자료형 데이터를 넣는 경우.
  - sum(iterable 자료형, 수치 자료형)

```
〉〉〉 L = [1,2,3,4,5]
〉〉〉 sum(L)              # 리스트 L의 합을 구합니다.
15
〉〉〉 sum(L, 10)          # 리스트 L의 원소들과 10의 합을 구합니다.
25
〉〉〉 sum(L, 10, 20)      # sum 함수는 인수를 세 개 가질 수 없습니다.
......
TypeError: sum expected at most 2 arguments, got 3
〉〉〉 sum(10, 20)         # sum 함수의 첫 번째 인수는 iterable 자료형이어야 합니다.
......
TypeError: 'int' object is not iterable
〉〉〉 M = [10, 20]
〉〉〉 sum(L, M)           # sum의 두 번째 인수는 수치 자료형이어야 합니다.
......
TypeError: can only concatenate list (not "int") to list
```

2. 에러 종류

- TypeError
  - for 반복문에 iterable 자료형을 넣지 않은 경우.

```
>>> for x in 10000:          # in 다음에는 반드시 iterable 객체가 와야 합니다.
        print(x)
.....
TypeError: 'int' object is not iterable
```

2. 에러 종류

- TypeError
  - 함수 호출시에 인수의 자료형을 결정하기 때문에 다음과 같은 경우에도 TypeError를 조심해야 함.

코드 1

정수 정수

```
def add_two(a, b):
    total = a + b
    return total

# main

x, y = 10, 20
result = add_two(x, y)
print(result)
```

결과 1

```
300
```

코드 2

문자열 문자열

```
def add_two(a, b):
    total = a + b
    return total

# main

x, y = 'hello', 'world'
result = add_two(x, y)
print(result)
```

결과 2

```
helloworld
```

코드 3

문자열 정수

```
def add_two(a, b):
    total = a + b
    return total

# main

x, y = 'hello', 200
result = add_two(x, y)
print(result)
```

결과 3

```
TypeError 발생
```

2. 에러 종류

- NameError - 없는 변수, 함수 등을 사용하려고 할 때.

```
>>> a = 10; b = 20
>>> d = a + b + c                # c는 없는 변수입니다.
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    d = a + b + c
NameError: name 'c' is not defined
>>> del a
>>> print(a)
Traceback (most recent call last):
  File "<pyshell#21>", line 1, in <module>
    print(a)
NameError: name 'a' is not defined
```

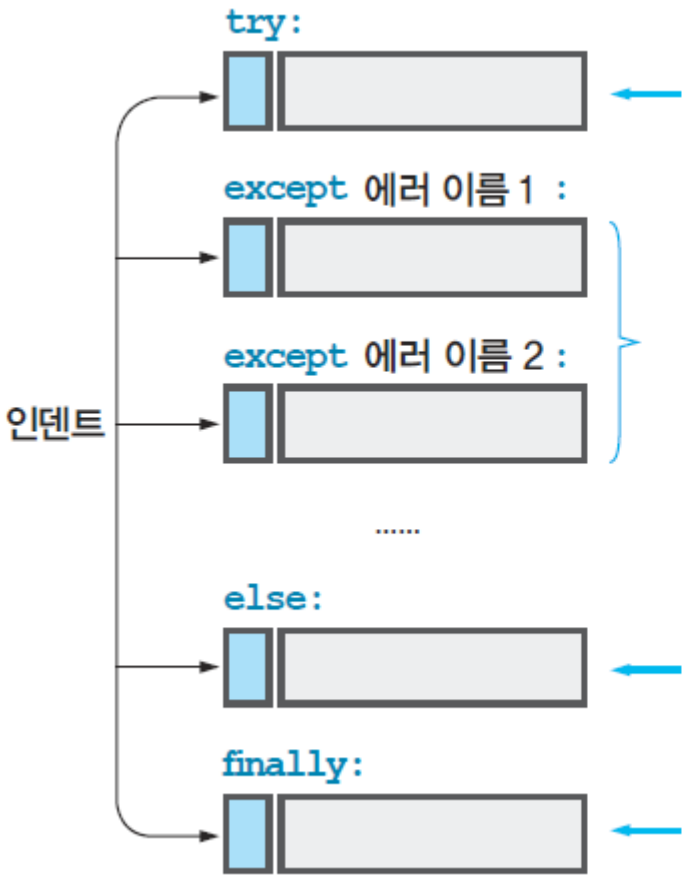
3. 예외 처리 기초

◆ 예외 발생 가능한 코드

코드	실행 ①	실행 ②	실행 ③
❶ a = int(input('a : ')) ❷ b = int(input('b : ')) ❸ c = a / b ❹ print(c) ❺ print('good bye')	a : 10 b : 5 2.0 good bye.	a : 0 b : 10 0.0 good bye	a : 5 b : 0 ..... ZeroDivisionError: .....

3. 예외 처리 기초

◆ 예외 처리 문법



'try 블록'에는 에러가 발생할 가능성이 있는 코드를 넣습니다. 한 줄씩 수행되다가 에러가 발생하면, try 블록을 빠져 나갑니다. 만약에 에러가 발생하지 않으면, try 블록을 모두 수행합니다.

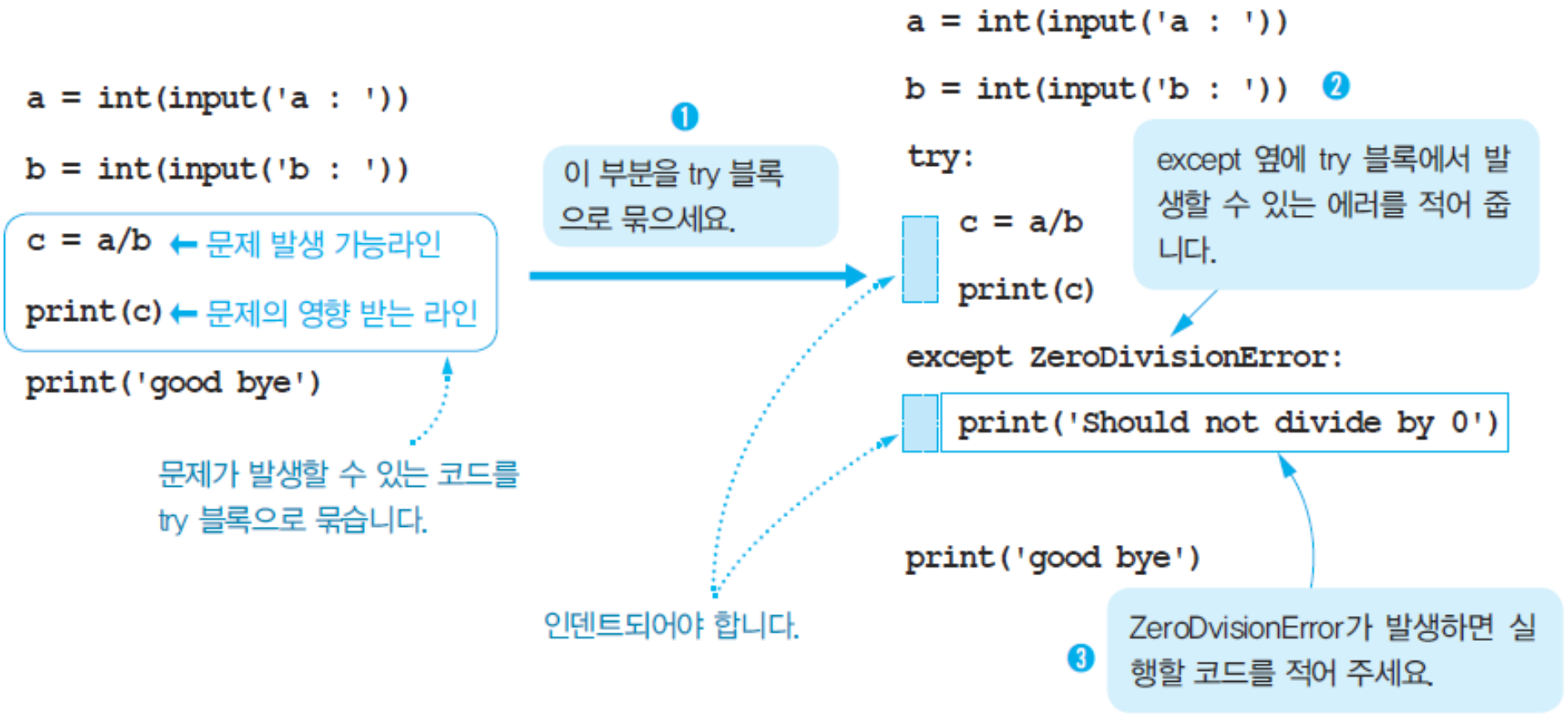
'except 블록'은 여러 개일 수 있습니다. except 키워드 옆에는 try 블록에서 발생 가능한 에러들을 적어 줍니다. try 블록에서 에러가 발생하지 않으면, except 블록은 모두 무시합니다. try 블록에서 에러가 발생하는 경우에 발생한 에러가 적힌 except 블록을 수행합니다. except 블록은 여러 개가 있더라도 해당 에러 부분만 수행됩니다.

'else 블록'은 있을 수도 있고, 없는 경우도 있습니다. else 블록이 있다면, try 블록에 에러가 없는 경우에만 수행됩니다.

'finally 블록'은 있을 수도 있고, 없는 경우도 있습니다. finally 블록이 있다면, 에러 발생과 상관없이 무조건 수행됩니다.

3. 예외 처리 기초

◆ 예외 처리 문법



3. 예외 처리 기초

◆ 예외 처리 문법

```
a = int(input('a : '))
b = int(input('b : '))

try:
    c = a/b
    print(c)
except ZeroDivisionError:
    print('Should not divide by 0')

print('good bye')
```

① a를 입력받습니다.

② b를 입력받습니다.

③ try를 만나면 try 블록의 첫 줄부터 수행합니다.

④ b가 0인 경우와 아닌 경우에 따라 수행이 다릅니다.

- **경우 1** b가 0이 아닌 경우
  아무 문제없이 c=a/b가 수행되고 print(c)도 수행됩니다. try 블록 안에서 아무 문제가 없기 때문에 except는 건너뜁니다.
- **경우 2** b가 0인 경우
  c=a/b에서 ZeroDivisionError가 발생하게 되고 try 블록을 빠져나가서 except 중에 ZeroDivisionError가 있는지 살펴 봅니다.

⑤ 위의 **경우 2** 일때, except block이 수행되어 'should not divide by 0'가 출력됩니다.

⑥ 'good bye'는 에러 발생과 무관하게 항상 출력됩니다.

try 블록, except 블록, 인덴트, 에러와 관계없는 문장입니다.

3. 예외 처리 기초

◆ 코드에 에러가 한 개인 경우의 예외 처리

```
fruits = ['apple', 'melon', 'orange']
your_favorite = input('What is your favorite fruit : ')
x = fruits.index(your_favorite)
print('{} is my number {} favorite fruit.'.format(your_favorite, x+1))
```

정상 수행	What is your favorite fruit : melon melon is my number 2 favorite fruit.
정상 수행	What is your favorite fruit : apple apple is my number 1 favorite fruit.
정상 수행	What is your favorite fruit : orange orange is my number 3 favorite fruit.
ValueError 발생	What is your favorite fruit : kiwi ..... ValueError: 'kiwi' is not in list

3. 예외 처리 기초

◆ 앞 코드에 ValueError 처리하기

```
fruits = ['apple', 'melon', 'orange']
your_favorite = input('What is your favorite fruit : ')
try:
    x = fruits.index(your_favorite)
    print('{} is my number {} favorite fruit.'.format(your_favorite, x+1))
except ValueError:
    print('I don₩'t like {} much.'.format(your_favorite))
else:
    print('We both like {}'.format(your_favorite))
```

정상 수행	What is your favorite fruit : melon melon is my number 2 favorite fruit.
ValueError 발생	What is your favorite fruit : strawberry I don't like strawberry much.

3. 예외 처리 기초

◆ 앞 코드에 ValueError 대신 KeyError로 처리하기

```
fruits = ['apple', 'melon', 'orange']
your_favorite = input('What is your favorite fruit : ')
try:
    x = fruits.index(your_favorite)
    print('{} is my number {} favorite fruit.'.format(your_favorite, x+1))
except KeyError:    # ValueError를 KeyError로 바꿈.
    print('I don₩'t like {} much.'.format(your_favorite))
else:
    print('We both like {}'.format(your_favorite))
```

에러 발생	What is your favorite fruit : kiwi ..... ValueError: 'kiwi' is not in list

3. 예외 처리 기초

◆ 코드에 에러가 여러 개인 경우의 예외 처리

▪ 에러가 여러 개인 코드

```
    a = int(input('Enter a : '))
    b = int(input('Enter b : '))
    L = [1,3,5]
① c = a / b
② print(c)
③ L[3] = 100
④ print(L)
    print('End of the program')
```

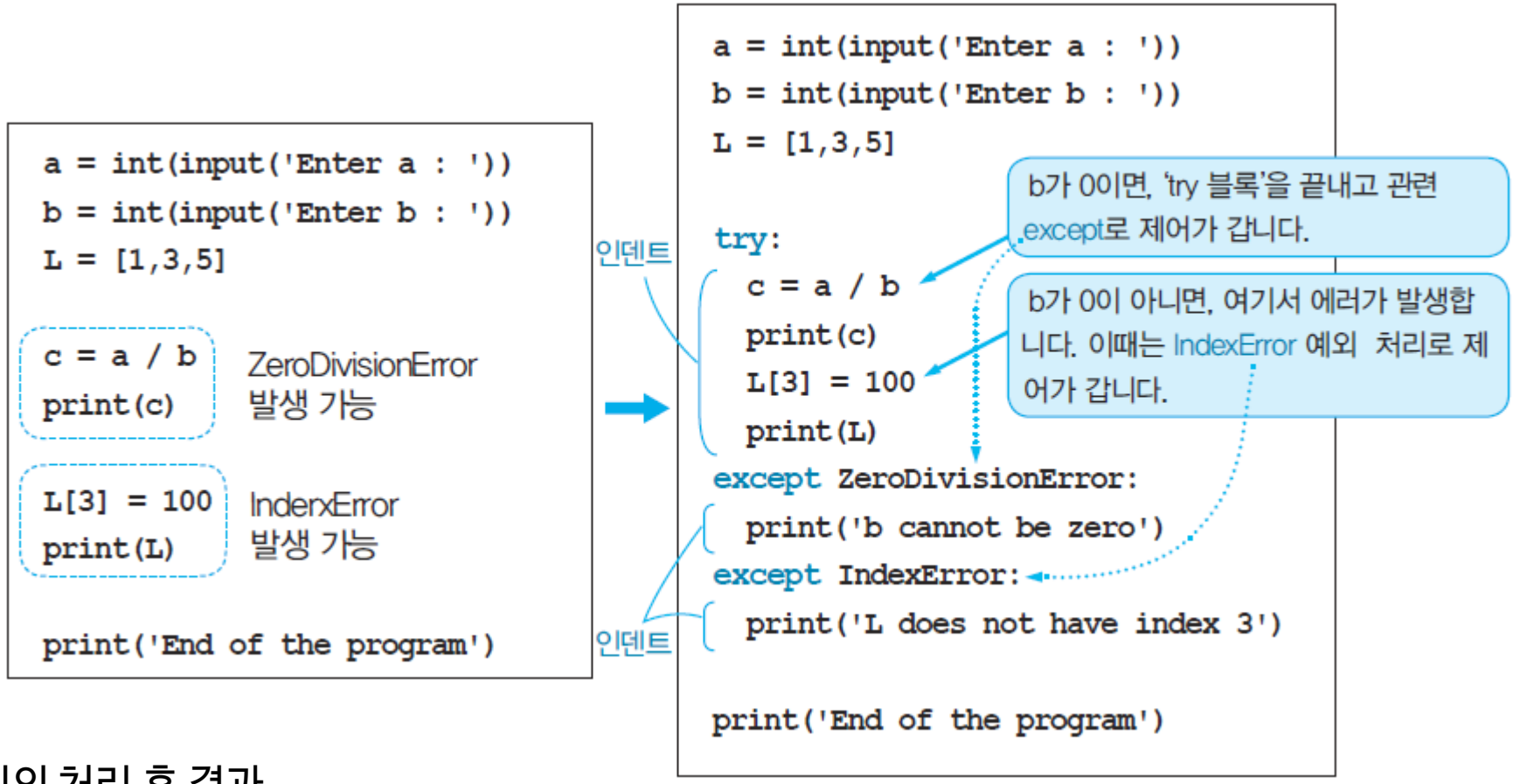
[결과 1]
① 에서 ZeroDivisionError 발생

```
Enter a : 10
Enter b : 0
......
ZeroDivisionError: division by zero
```

[결과 2]
③ 에서 IndexError 발생

```
Enter a : 10
Enter b : 5
2.0
......
IndexError: list assignment index out of range
```

3. 예외 처리 기초



예외 처리 후 결과

Enter a : 10 Enter b : 0 b cannot be zero End of the program	Enter a : 10 Enter b : 5 2.0 L does not have index 3 End of the program

3. 예외 처리 기초

◆ except 옆에 아무 에러도 적지 않는 경우

<table>
<tr><td rowspan="2">IndexError</td><td>[IndexError 처리 전 코드]

L = [23, 29, 17]
print(L[1])
print(L[3]) # 에러
print(L[-3])</td><td>[IndexError 처리 후 코드]
L = [23, 29, 17]
print(L[1])
try:
  print(L[3])
except: # except만 적음.
  print('Check index!')
print(L[-3])</td></tr>
<tr><td>[결과]

29
.....
IndexError: .....</td><td>[결과]

29
Check index!
23</td></tr>
</table>

3. 예외 처리 기초

◆ except 옆에 아무 에러도 적지 않는 경우

<table>
<tr><td rowspan="2">TypeError</td><td>[TypeError 처리 전 코드]

L = [1, 3, 5]
T = (7, 9)
V = L + T # 에러
print(V)</td><td>[TypeError 처리 후 코드]

L = [1, 3, 5]
T = (7, 9)
try:
  V = L + T
  print(V)
except: # except만 적음.
  print('Check the type!')</td></tr>
<tr><td>[결과]

.....
TypeError: .....</td><td>[결과]

Check the type!</td></tr>
</table>

3. 예외 처리 기초

◆ except 옆에 아무 에러도 적지 않는 경우

<table>
<tr><td rowspan="2">ZeroDevisionError</td><td>[ZeroDivisionError 처리 전 코드]

a = 10
b = 0
c = a / b # 에러
print(c)</td><td>[ZeroDivisionError 처리 후 코드]

a = 10
b = 0
try:
  c = a / b
  print(c)
except: # except만 적음.
  print('cannot divide by zero')</td></tr>
<tr><td>[결과]

.....
ZeroDivisionError: .....</td><td>[결과]

cannot divide by zero</td></tr>
</table>

3. 예외 처리 기초

- except 옆에 Exception이라고 적기도 함.

```
a = int(input('Enter a : '))
b = int(input('Enter b : '))
L = [1, 3, 5]
try:
    c = a / b            # b가 0이면 ZeroDivisionError가 발생합니다.
    print(c)
    L[3] = 100           # IndexError가 발생합니다.
    print(L)
except Exception:    # try 블록에서 어떤 에러가 발생하든지 예외 처리를 하겠다는 표현입니다.
    print('Some error occured')
```

[결과 1]

ZeroDivisionError가 발생한 경우
Enter a : 5
Enter b : 0
Some error occured

[결과 2]

IndexError가 발생한 경우
Enter a : 5
Enter b : 1
5.0
Some error occured

3. 예외 처리 기초

◆ except ~ as 구문

```
try :
   < 에러 발생 가능성이 있는 문장 >
except 에러 이름 as e :      # e는 변수예요. 다른 이름으로 해도 됩니다.
   < 예외 처리 문장 >            에러가 발생했을 때 어떤 에러인지 e에 저장됩니다.
....
else :
   < 예외가 발생하지 않은 경우에 수행할 문장 >
finally :
   < 예외 발생 유무에 상관없이 try 블록 이후 수행할 문장 >
```

3. 예외 처리 기초

◆ except ~ as 구문

```
x = int(input('x : '))
y = input('y : ')
try:
    z = x + y          # 정수 + 문자열은 TypeError가 발생합니다.
except TypeError as e:
    print(e)
```

[결과]

```
x : 5
y : 7
unsupported operand type(s) for +: 'int' and 'str'
```

print(e)의 결과

```
>>> x = 5; y = '7'
>>> z = x + y
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    z = x + y
TypeError: unsupported operand type(s) for +: 'int' and 'str'   # 에러 메시지
```

이내용이 except TypeError as e에서 e에 들어감.

3. 예외 처리 기초

- except ~ as 구문

[코드 1]	[코드 2]

L = [90, 88, 98, 70]
try:
  score = L[5]
  print(score)
except IndexError as err:
  print(err)
	L = [90, 88, 98, 70]
	try:
	  score = L.index(100)
	  print(score)
	except ValueError as err:
	  print(err)

[결과 1] list index out of range	[결과 2] 100 is not in list
[셸에서 보는 에러 메시지] 〉〉〉 L = [90, 88, 98, 70] 〉〉〉 score = L[5] ..... IndexError: list index out of range	[셸에서 보는 에러 메시지] 〉〉〉 L = [90, 88, 98, 70] 〉〉〉 score = L.index(100) ..... ValueError: 100 is not in list

3. 예외 처리 기초

◆ 에러가 여러 개인 경우 Exception 사용시 주의점

```
a = int(input('Enter a : '))
b = int(input('Enter b : '))
L = [1,3,5]
try:
    c = a / b      # b에 0을 넣는다면...
    print(c)
    L[3] = 100
    print(L)
except Exception:
    print('L does not have index 3')
except ZeroDivisionError:
    print('b cannot be zero')
print('End of the program')
```

[결과 1]
b에 0을 넣으면 c = a / b에서 ZeroDivisionError가 발생함.
그리고 첫 번째 except로 가서 Exception에 해당하는 에러인지를 판단함. Exception은 어떤 에러도 처리해 주기 때문에 다음에 있는 ZeroDivisionError까지 가지 않음. 그래서 제대로 예외 처리를 못하는 경우가 생김.

[결과 2]

```
Enter a : 10
Enter b : 0
L does not have index 3
End of the program
```

3. 예외 처리 기초

◆ except : pass 구문

- 에러가 발생했을 때, 특별히 처리할 내용이 없다면 간단히 pass로 블록을 완성할 수 있음.

```
a = int(input('a : '))
b = int(input('b : '))
L = [4, 8, 12]
try:
    c = a / b              # ZeroDivisionError
    v = L.index(3)         # IndexError
except:
    pass
finally:
    print('finally block entered')
```

[결과 1]

```
a : 10
b : 0
finally block entered
```

[결과 2]

```
a : 10
b : 5
finally block entered
```

4. else 블록 이해하기

- try 구문에서 else 블록은 있기도 하고 없기도 함.
  - else 블록이 있는 경우, 에러가 발생하지 않으면 수행됨.
  - else 블록이 있어도 에러가 발생하면 else 구문은 수행되지 않음.

예외 발생하는 코드	예외 발생하지 않는 코드

L = [1,3,5]
try:
  L[3] = 100 # IndexError 발생
  print(L)
except IndexError:
  print('L does not have index 3')
else:
  print('exception not occurred')
	L = [1,3,5]
	try: # try 블록에 에러 없음
	  L[2] = 100
	  print(L)
	except IndexError:
	  print('L does not have index 3')
	else:
	  print('I am else')

[결과] L does not have index 3	[결과] [1, 3, 100] I am else

5. finally 블록 이해하기

- finally 블록은 try 구문에 있기도 하고 없기도 함.
  - finally 블록은 무조건 수행되는 구문임.

```
L = [1, 2, 3, 4]
x = int(input('Enter data to remove : '))
try:
    L.remove(x)
except ValueError:
    print('Error : index out of range')
else:
    print('exception not occurred')
finally:
    print('try ends here')
```

[결과 1]

```
Enter data to remove : 3
exception not occurred
try ends here
```

[결과 2]

```
Enter data to remove : 10
Error : index out of range
try ends here
```

5. finally 블록 이해하기

◆ try ~ finally 만 있는 구문 (except가 없음)

```
a = int(input('Enter a : '))
b = int(input('Enter b : '))
L = [1,3,5]
try:
    c = a / b
    print(c)
    L[3] = 100
    print(L)
finally:
    print('finally statements')
```

- try 구문이 있으면, 에러는 찾음.
- except 블록이 없기 때문에 에러 처리를 못함.
- finally 구문은 무조건 수행됨.

```
Enter a : 5
Enter b : 0
finally statements
Traceback (most recent call last):
......
  c = a / b
ZeroDivisionError: division by zero
```

```
Enter a : 10
Enter b : 5
2.0
finally statements
Traceback (most recent call last):
......
  L[3] = 100
IndexError: list assignment index out of range
```

6. 예외를 직접 발생시키기 – raise

◆ 프로그래머가 필요할 때 에러 발생시키기 – raise

```
print('start of the program')
try:
    print('start of try block')
    raise ValueError        # ValueError 발생시킴
    print('cannot reach this line')
except ValueError:
    print('exception occurred')
```

[결과]

```
start of the program
start of try block
exception occurred
```

6. 예외를 직접 발생시키기 – raise

- raise 예제

```
answer = input("Enter y or n to continue : ")
try:
    if answer == 'y':
        print("Let's continue ~ ")
    elif answer == 'n':
        print("Let's stop ~ ")
    else:
        raise ValueError     # 에러 발생시킴.
except ValueError:
    print(answer, 'is not acceptable answer')
```

[결과]

```
Enter y or n to continue : y
Let's continue ~

Enter y or n to continue : n
Let's stop ~

Enter y or n to continue : t
t is not acceptable answer
```

7. 정리

- 파이썬에서는 문법 에러인 SyntaxError와 다른 여러 에러들이 발생할 수 있음.
- 문법 에러인 SyntaxError는 수정해야 하지만, 다른 에러들은 예외처리를 통해서 에러를 해결하도록 권장함. ('EAFP' 코딩 스타일)
- 예외 처리는 try ~ except ~ finally 구문을 이용함.