파이썬 프로그래밍

13. 예외 처리, 문자열, 람다 함수, map 함수

00

Contents

❖ 수업 목표

- 예외 처리 코드를 작성할 수 있다.
- 다양한 문자열 처리 코드를 작성할 수 있다.
- 람다 함수를 작성할 수 있다.
- map 함수를 작성할 수 있다.

❖ 세부 목표

00

1. 예외 처리

❖ 에러와 예외

▪ 문법 에러

- 프로그래밍 언어 작성 방법 오류
- 오류 발생 내용 확인 가능
- 실행 중단

```
>>> while True print('Hello world')

  File "<stdin>", line 1
    while True print('Hello world')
               ^^^^^
SyntaxError: invalid syntax
```

1. 예외 처리

❖ 에러와 예외

- 예외
  - 문법적으로 문제가 없으나 실행 과정에서 발생
  - 예외 처리를 통해 실행 상태 유지 가능

- 예외 예시
  - ZeroDivisionError
    - >>> 10 * (1/0)
    - Traceback (most recent call last):
    - File "<stdin>", line 1, in <module>
    - ZeroDivisionError: division by zero

1. 예외 처리

❖ 에러와 예외

▪ 예외 예시

- **NameError**
  - `>>> 4 + spam*3`
  - `Traceback (most recent call last):`
  - `  File "<stdin>", line 1, in <module>`
  - `NameError: name 'spam' is not defined`

- **TypeError**
  - `>>> '2' + 2`
  - `Traceback (most recent call last):`
  - `  File "<stdin>", line 1, in <module>`
  - `TypeError: can only concatenate str (not "int") to str`

1. 예외 처리

❖ 예외 처리하기

- 선택한 예외를 처리하는 프로그램을 만드는 것이 가능
- try ... except 구문
  - 먼저, try 절 실행
  - 예외 미 발생시,
    - except 절 을 건너뛰고 try 문의 실행은 종료
  - 예외 발생 시,
    - try나머지 절은 건너뜀
    - except 절에 명시된 예외와 예외 유형이 일치 시, except 절 실행
    - except 절에 명시된 예외와 예외 유형이 불일치 시, 실행 중지

1. 예외 처리

❖ 예외 처리하기

▪ try ... except 구문 예시

- 올바른 정수가 입력될 때까지 사용자에게 입력 요청 프로그램

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

1. 예외 처리

❖ 예외 처리하기

▪ try ... except 구문

- try은 다른 예외의 처리를 위한 여러 개의 except 절을 가질 수 있음
  - 최대 하나의 핸들러만 실행

- except 절은 여러 예외를 괄호로 묶은 튜플로 명명 가능
  - 예) except (RuntimeError, TypeError, NameError):

- 각 예외에는 예외 관련 구체적 정보가 저장됨
- 예외 정보는 except 절의 예외 이름 뒤 변수를 as로 지정하여 확인 가능

```
except Exception as e:
    print(e)
```

1. 예외 처리

❖ 예외 클래스

- 모든 예외의 공통 기본 클래스: BaseException
- 실행 중 감지되는 예외의 기본 클래스: Exception
  - 시스템 종료 외 모든 예외 처리하는 용도로 사용 가능
  - 단, 예외 유형은 구체적으로 지정하는 것을 권장

```
try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error:", err)
except ValueError:
    print("Could not convert data to an integer.")
except Exception as err:
    print(f"Unexpected {err=}, {type(err)=}")
```

1. 예외 처리

❖ 예외 일으키기

▪ raise 문

- 프로그래머가 지정한 예외가 발생하도록 강제
- 예외 클래스 중 하나로 지정

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')
```

```
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
NameError: HiThere
```

2. 문자열 처리

❖ 문자열 이해

- 제시된 프로그램을 실행

```
muna="python"
print(muna[0])
print(muna[1])
print(muna[2])
print(type(muna))
muna[0]='k' # 문자열은 문자열 상수이다. 상수는 변경할 수 없어 아래 오류가 발생한다.
# TypeError: 'str' object does not support item assignment

munb=["python"]
print(munb[0])  # 문자열이 하나의 요소(item)이다.
print(type(munb))

munc=["p","y","t","h","o","n"]
print(munc[0])
print(munc[1])
print(munc[2])
print(type(munc))
munc[0]='k'
print(munc)
```

2. 문자열 처리

❖ 문자열 이해

▪ for 문으로 문자 출력

- 제시된 프로그램의 munc 변수에 저장된 문자를 for 문을 이용하여 출력

```
for i in range(0, 6, 1):
    print(munc[i])
```

2. 문자열 처리

❖ 문자열 이해

▪ 문자열 길이 구하기

- 문자열 수를 지정하지 않고 프로그램에서 길이를 구한 다음 출력

내장 함수 : len("문자열")

문자열의 길이를 리턴한다.

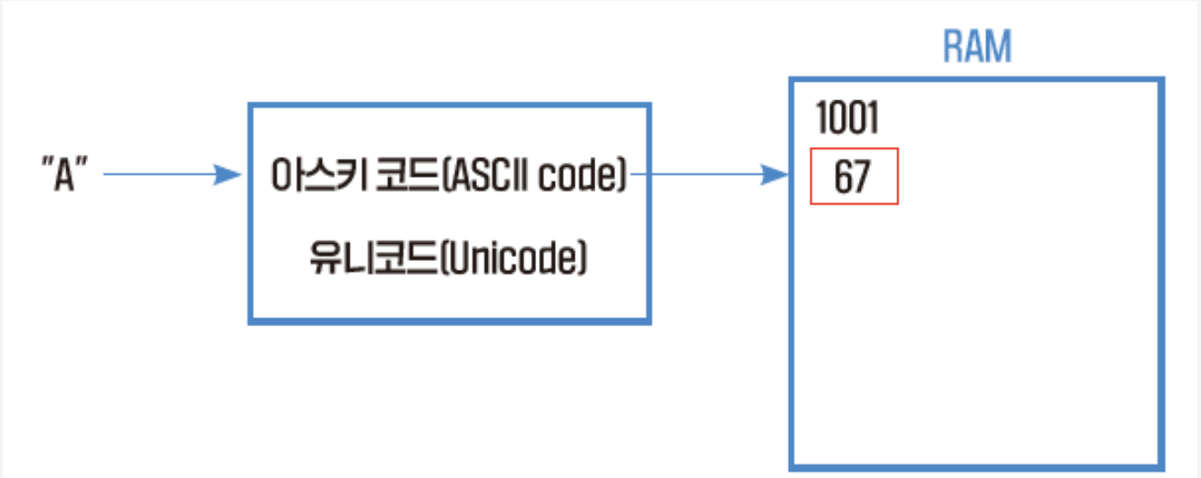
```
length=len(munc)
print(length)

for i in range(0, length, 1):
    print(munc[i])
```

2. 문자열 처리

❖ 문자열은 어떻게 주기억장치에 저장되는 것일까?

- 아스키 코드(American Standard Code for Information Interchange, ASCII)
  - 컴퓨터에 데이터 저장 시, 0과 1을 이용하여 숫자를 표현하는 방식
  - 문자를 저장하기 위해 각 문자를 특정 숫자로 표현하는 약속
  - 영문자 이외의 문자를 표현하기 위해서 다른 형식의 약속이 추가로 필요

- 유니코드(Unicode)
  - 세계 다양한 언어의 문자를 표현하기 위해 확장된 대표적인 약속(규정)



2. 문자열 처리

❖ 문자열 이해

- 컴퓨터에 저장되는 문자를 이해하기 위해 파이썬에서 제공하는 내장 함수

내장 함수	설명	예시
ord(인수)	한 글자로 된 문자를 인자로 받아 이 문자의 컴퓨터 표현에 해당하는 숫자를 리턴한다.	ord("a")
chr(인수)	숫자를 인자로 받아 이 숫자가 나타내는 문자를 리턴한다.	chr(97)

2. 문자열 처리

❖ 문자열 연산자

▪ + 연산자

- 피연산자의 데이터형이 문자열인 경우, 문자열 연결 연산 수행

syntax : 문자열 연결하기

문자열 + 문자열	ma = "ChatGPT" + "를 활용한 python" print(ma)

2. 문자열 처리

❖ 문자열 출력 방법: 형식지정자

- 형식지정자 뒤에 ()와 변수 값을 넣어주는 형태
- % 연산자 사용 방식

syntax : 형식지정자 사용

```
형식지정자 자료형
%s 문자열 String(문자열)
%d 정수 Decimal(십진수, 정수)
%f 실수 float(실수)
```

2. 문자열 처리

❖ 문자열 출력 방법: f-string

- 서식 문자열의 중간에 들어가는 각 변수들은 {변수}를 사용하여 표시
- 문자열 시작을 나타내는 따옴표 앞에 ‘f ’ 문자를 입력

syntax : 문자열 포맷(string format), f-string

```
print(f'문자열식:{변수명}으로 표현 가능')
```

2. 문자열 처리

❖ 문자열 출력 방법 예시

na 값 20 더하기 nb 값 5 의 결괏값은 25 이다.

```
print("na 값",na,"더하기 nb 값",nb,"의 결괏값은",na+nb,"이다.")
```

```
print("na 값 %d 더하기 nb 값 %d 의 결괏값은 %d 이다."%(na,nb,na+nb))
```

```
print(f"na 값 {na} 더하기 nb 값 {nb} 의 결괏값은 {na+nb} 이다.")
```

2. 문자열 처리

❖ 인덱싱

- 리스트 선언 시, 자동으로 0부터 인덱스 생성
- 인덱스를 이용하여 특정한 데이터에 접근하는 과정
  - 왼쪽에서 시작하는 인덱스는 0부터 시작
  - 오른쪽(데이터 가장 끝)부터 시작하는 인덱스는 –1부터 시작
    - -1은 가장 오른쪽(뒤에서 첫 번째) 요소를 가리킴
    - -2는 뒤에서 두 번째 요소를 가리킴

syntax : 인덱싱

변수명[인덱스]	muna="python" print(muna[0]) print(muna[-1])

2. 문자열 처리

❖ 슬라이싱

- 시퀀스 자료형(문자열, 리스트, 튜플 등)의 부분 집합을 추출하는 방법
- 시퀀스 객체에서 중간 일부를 추출할 때 사용
- [ ]를 사용하며, 인덱스와 콜론(:)을 이용하여 추출

syntax : 슬라이싱

변수명[인덱스 : 인덱스]	muna="python" print(muna[2:-1]) # ----> ① print(muna[1:4]) # -----> ② print(muna[:]) # -------> ③

- ① muna[2:-1]은 문자열 인덱스 2부터 인덱스 (-1-1)까지의 부분 추출
- ② muna[1:4]는 문자열 인덱스 1부터 인덱스 (4-1)까지의 부분 추출
- ③ muna[:]은 문자열 muna 전체를 추출

3. 문자열 함수

❖ split( )

- 문자열을 특정 구분자(Separator)를 기준으로 분리하여 리스트로 반환
- 구분자는 인자로 전달되며, 인자를 지정하지 않으면 공백을 구분자로 사용

syntax : split()

```
문자열.split(구분자)
```

3. 문자열 함수

❖ split( )

▪ split( ) 코드

```
my_string="Python is a popular programming language"
split_list=my_string.split()
print(split_list)
```

〈화면 출력〉

```
['Python', 'is', 'a', 'popular', 'programming', 'language']
```

- 문자열을 공백을 기준으로 분리하여 리스트 형태로 반환 가능

3. 문자열 함수

❖ strip( )

- 문자열의 양 끝에 있는 공백(띄어쓰기, 탭, 개행문자 등)을 제거한 문자열을 반환

syntax : strip()

```
문자열.strip()
```

3. 문자열 함수

❖ strip( )

▪ strip( ) 코드

- 문자열을 선언하고 strip( ) 메소드를 사용하면, 양 끝의 공백이 제거된 문자열을 반환

```
my_string="   Python is awesome!   "
stripped_string=my_string.strip()
print(stripped_string)
```

〈화면 출력〉

```
'Python is awesome!'
```

- 문자열의 양 끝에 있는 공백만을 제거하므로, 문자열 중간에 있는 공백은 제거되지 않는다.

3. 문자열 함수

❖ join()

- 리스트나 튜플과 같은 iterable 객체의 모든 요소를 하나의 문자열로 결합하여 반환
- 호출하는 문자열은 연결할 요소 사이에 삽입

syntax : join()

```
구분자.join(리스트)
```

3. 문자열 함수

❖ join()

▪ join() 코드

- 리스트를 선언하고 join( ) 메소드를 사용하면, 리스트의 모든 요소가 하나의 문자열로 결합되어 반환

```
my_list=["apple", "banana", "cherry"]
joined_string="-".join(my_list)
print(joined_string)
```

〈화면 출력〉

```
'apple-banana-cherry'
```

- 호출하는 문자열인 “-” 은 리스트 요소들을 연결하는 구분자 역할
- 구분자를 지정하지 않고 join( ) 메소드를 호출하면, 리스트 요소들이 연속적으로 결합

4. 람다 함수

❖ 람다(Lambda) 함수

- 이름이 지정되지 않은 함수(Anonymous Function)
- def 키워드를 사용하지 않고도 함수를 정의 가능

syntax : lambda 함수

lambda 인자 : 표현식	lambda x : x+x

4. 람다 함수

lambda 키워드

- lambda 키워드를 사용하여 square 함수를 정의
- 이 함수를 호출하여 3의 제곱 값을 계산

```
square=lambda x : x ** 2
print(square(3))
```

〈화면 출력〉

```
9
```

5. map() 함수

❖ map() 함수

- 리스트나 튜플 등의 반복 가능한(Iterable) 객체의 각 요소에 대해 지정된 함수를 적용하여 그 결과를 새로운 이터레이터(Iterator)로 반환하는 함수

syntax : map() 함수

map(함수, 이터레이터)	map(square, numbers)

5. map() 함수

❖ map() 함수

- 리스트나 튜플 등의 반복 가능한(Iterable) 객체의 각 요소에 대해 지정된 함수를 적용하여 그 결과를 새로운 이터레이터(Iterator)로 반환하는 함수

5. map() 함수

❖ map( ) 코드

- numbers 리스트의 각 요소에 square( ) 함수를 적용
- 적용한 결과를 squared _numbers 리스트에 저장

```
def square(x):
    return x ** 2

numbers=[1, 2, 3, 4, 5]
squared_numbers=map(square, numbers)
print(list(squared_numbers))
```

〈화면 출력〉

```
[1, 4, 9, 16, 25]
```

5. map() 함수

❖ lambda 함수와 함께 사용

- 앞선 square 함수 호출 예제를 람다 함수와 map() 함수를 함께 사용하여 구현한 것

```
numbers=[1, 2, 3, 4, 5]
squared_numbers=map(lambda x: x ** 2, numbers)
print(list(squared_numbers))
```

〈화면 출력〉

```
[1, 4, 9, 16, 25]
```

00

Notice

❖ 과제

- 1. 다양한 예외 처리 코드 작성하기
- 2. 문자열 처리 코드 작성하기
- 3. 인덱싱과 슬라이싱 코드 작성하기
- 4. f-string 문자열 출력 코드 작성하기
- 5. map() 함수와 람다 함수 사용 코드 작성하기

❖ 다음 수업 내용

- 정규표현식
  - 정규표현식 문법, 메타문자, re모듈, 문자열 검색, match객체의 메서드, 컴파일 옵션, 역슬래시 문제