Chapter. 03

파이썬 프로그래밍 언어

파이썬 고급 주제들

- 예외 처리, 맵 리듀스, 파일입출력

FASTCAMPUS ONLINE

금융공학/퀀트

강사. 서찬웅

이번시간에 배울 내용

1. 기본 숫자 연산자
2. 비교 연산자
3. 논리 연산자
4. 연산자 우선순위

I 예외 처리에 대해서 알아보겠습니다.

- 예외처리는 프로그램이 무엇인가 잘못되었거나, 기대하지 않은 일이 발생 했을 때 알려주는 방법입니다.
- 모든 프로그램 언어는 예외 처리 구문이 존재합니다.
- 아래 예제는 list의 index 메소드를 사용할 경우 list 안에 값이 존재하지 않으면 오류(error)를 발생하는데 파이썬이 오류를 만나는 순간 오류의 내용을 출력하고 프로그램은 종료가 됩니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
stock.index('삼성SDS')
print ("이 문장은 실행이 되지 않습니다.")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-09468e9b332f> in <module>
      1 stock = ['삼성전자', '엘지전자', 'SK텔레콤']
----> 2 stock.index('삼성SDS')
      3 print ("이 문장은 실행이 되지 않습니다.")

ValueError: '삼성SDS' is not in list
```

- list의 index 메소드는 값이 없다면 Error가 발생한다. 삼성SDS가 없기 때문에 Error가 발생되고 print문은 실행이 되지 않는다.

I else 블록에 대해서 알아보겠습니다.

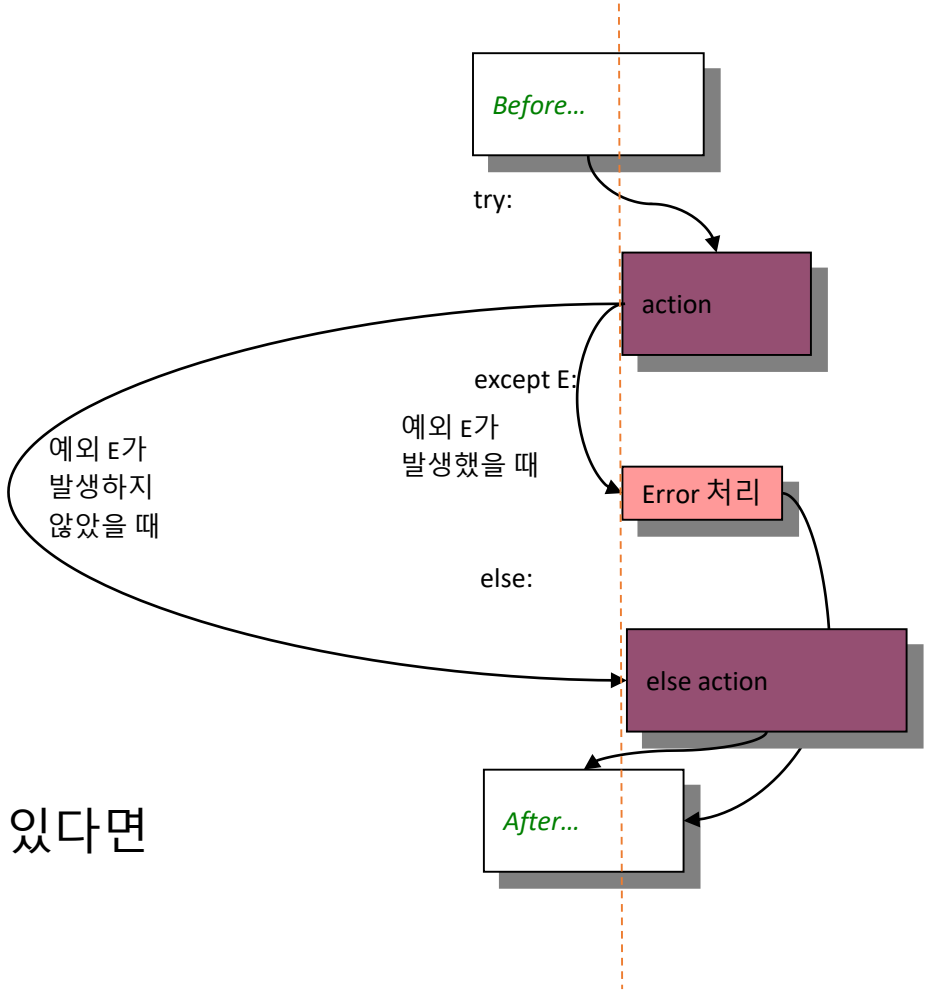
- Error가 발생 되었을 때 프로그램이 종료되지 않고 계속 실행이 될려면 프로그램에게 예외 발생 시에 어떻게 동작해야 할지를 알려주면 됩니다.
- 바로 그 구문이 try ~ except 블록입니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성SDS')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
```

```
삼성SDS가 없습니다.
```

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성전자')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
```

```
원하는 데이터가 있습니다.
```



- else 블록은 옵션입니다. 즉 없을수도 있고 있을수도 있습니다. 있다면 try 블록에서 Error가 발생되지 않을 때 실행이 됩니다.

I finally 블록에 대해서 알아보겠습니다.

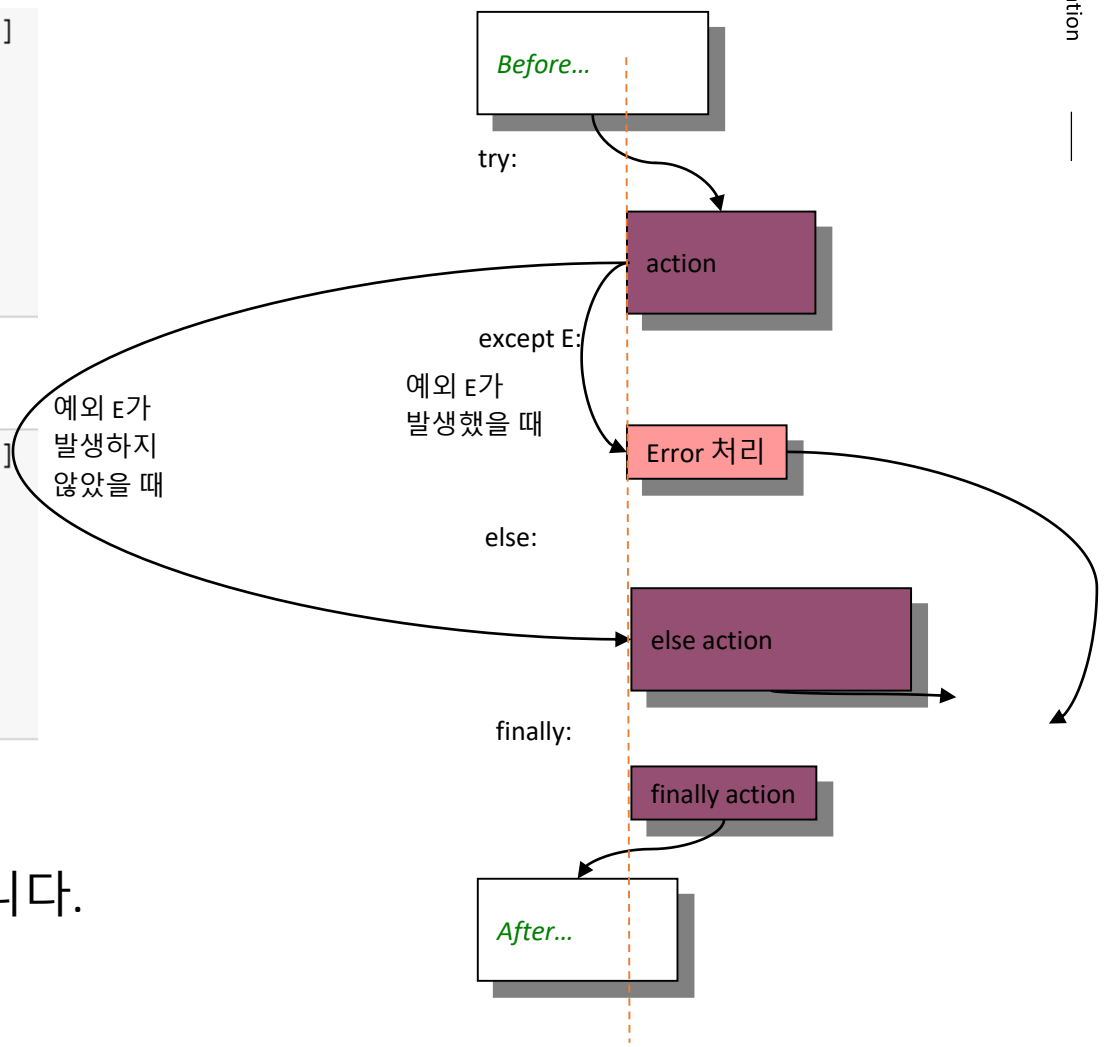
- finally 절은 예외 발생 여부에 상관없이 무조건 실행이 됩니다.
- 아래 예제에서 확인해보겠습니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성SDS')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
finally:
    print ("무조건 실행")
```

```
삼성SDS가 없습니다.
무조건 실행
```

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성전자')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
finally:
    print ("무조건 실행")
```

```
원하는 데이터가 있습니다.
무조건 실행
```



- finally 블록도 else블록과 마찬가지로 있을 수도 없을수도 있습니다.
  - 단 있다면 Error 발생 여부와 무조건 실행이 됩니다.

I Error의 종류 및 상황별 예외 처리 방법에 대해서 알아보겠습니다.

- Error의 종류는 아래와 같이 여러 종류가 존재합니다.
  - SyntaxError : 파이썬 문법을 잘못 사용해서 발생하는 Error입니다.
  - AttributeError : 모듈의 없는 속성(메소드)를 호출 했을 때 발생합니다.
  - IndexError : 해당 시퀀스에 인덱스 범위를 잘못 지정하였을 때 발생합니다.
  - KeyError : 집합(set)과 사전(dict)에서 발생합니다.
    - 집합에서는 없는 원소를 remove()하면 발생
    - 사전에는 없는 key 값을 호출하면 발생
  - ValueError : 어떤 값을 가져올 수 없는 경우 발생
  - FileNotFoundError : 존재하지 않은 파일을 open() 할 때 발생합니다.
  - TypeError : 자료형에 적합하지 않은 연산을 할 경우 발생
- 앞에서는 try ~ except 블록에서 except만을 사용하여 Error 처리를 진행하였습니다. except만 사용할 경우는 다양한 종류의 Error를 모두 except 블록에서 처리하겠다 의미입니다.
- 만약 Error 종류별로 상황별 처리를 하고 싶다면 except SyntaxError, except AttributeError... 사용하시면 됩니다.

I Error의 종류 및 상황별 예외 처리 방법에 대해서 알아보겠습니다.

```
try:
    5 / 0
except NameError:
    print ("NameError가 발생")
except IndexError:
    print ("IndexError가 발생")
except ZeroDivisionError:
    print ("ZeroDivisionError 발생")
```

```
ZeroDivisionError 발생
```

```
try:
    stock[3]
except NameError:
    print ("NameError가 발생")
except IndexError:
    print ("IndexError가 발생")
except ZeroDivisionError:
    print ('ZeroDivisionError 발생')
```

```
IndexError가 발생
```

Error의 종류에 따라서 해당 Error를 처리할 수 있다

FAST CAMPUS
ONLINE
서찬웅 강사.

I Error 구체적인 내용을 확인하는 방법을 알아보겠습니다.

- 예외가 발생하였을 때 발생된 Error의 내용을 확인하고 싶다면 except Exception as e 구문을 사용하시면 됩니다.

```
try:
    5 / 0
except Exception as e:
    print (e)
```

```
division by zero
```

- except 옆에 Error 종류를 적는 위치에 Exception만 적어도 except만 적는거와 동일한 결과가 발생됩니다.
  - except Exception:
  - except:
- except Error as e 표현식을 사용하면 Error의 구체적인 내용을 확인할 수 있습니다.
- 위의 예제처럼 except Exception as e 라고 표현하면 전체 Error에 대해서 해당되고 e 부분에는 구체적인 오류 메시지가 들어갑니다. 위의 예제에서 ZeroDivisionError만 처리하고 싶다면 except ZeroDivisionError as e :
  - as 키워드는 alias를 만들어주는 키워드이기 때문에 e 대신 사용자가 사용하고 싶은 단어를 쓰셔도 됩니다.

I pass에 대해서 알아보겠습니다.

- 아래 그림은 try ~ except ~ finally 전체 순서도입니다.
- 여기서 만약 excep 구문에 아무 작업도 하고 싶지 않다면 어떻게 해야 할까요?
- 그럴 때 사용하는게 pass 키워드입니다.
- pass 키워드를 만나는 순간 아무 작업도 하지 않고 키워드 그대로 pass 합니다.
- try ~ except 구문 뿐만 아니라 for문에서도 자주 사용됩니다.

```
for x in range(1,10):
    if x == 2:
        pass
        print ("After")
    print ("hmmm")
```

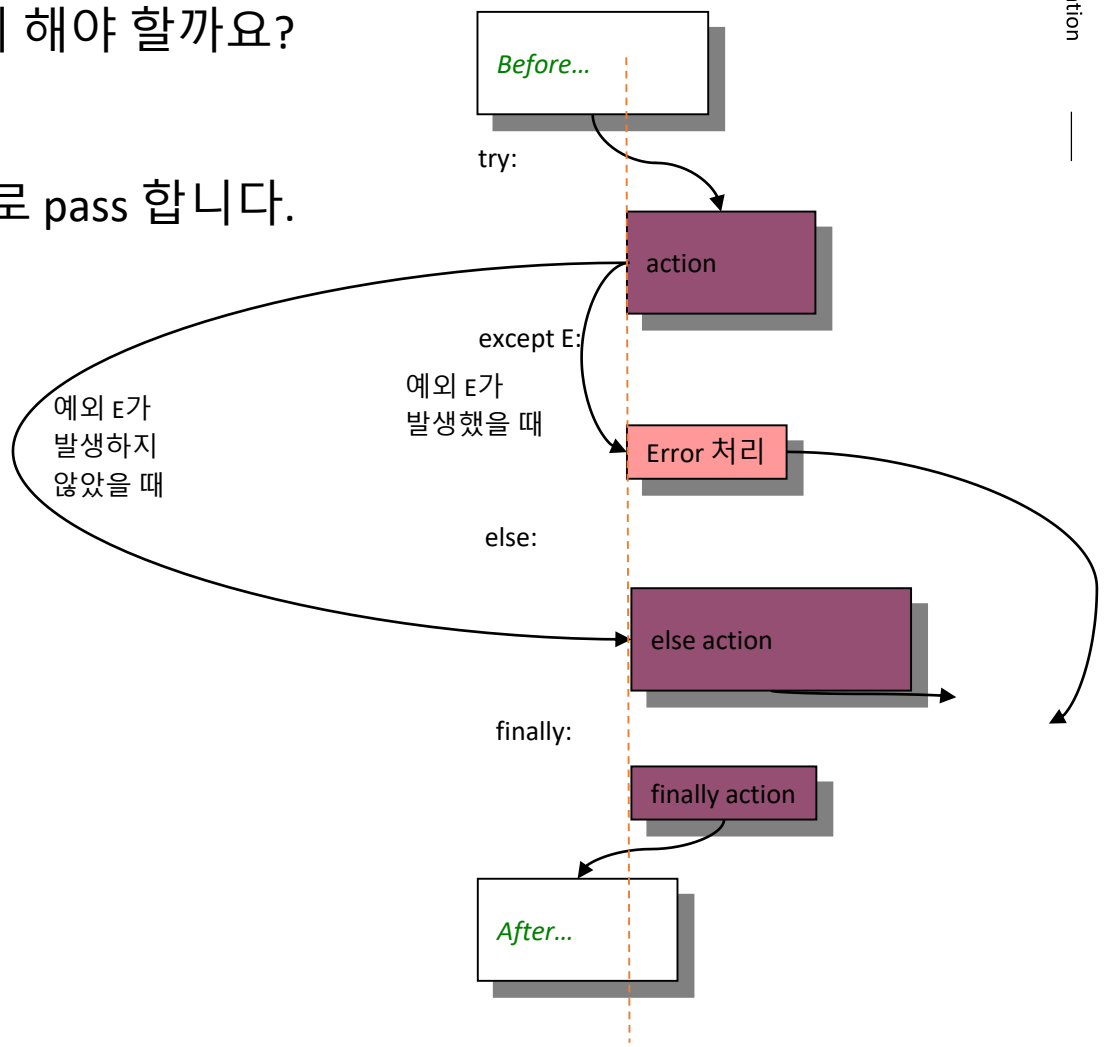
pass를 만나고 밑의 코드가 실행

```
hmmm
After
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
```

```
for x in range(1,10):
    if x == 2:
        continue
        print ("After")
    print ("hmmm")
```

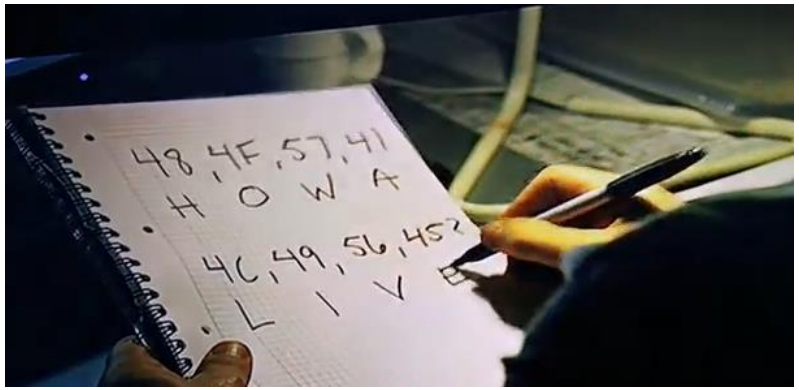
continue를 만나고 다음 for문 실행

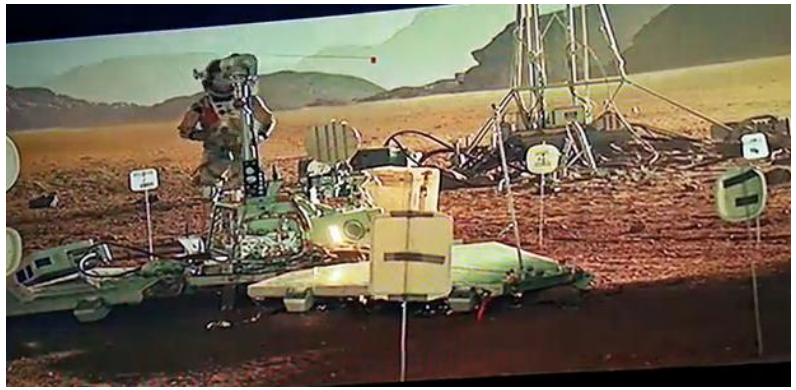
```
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
hmmm
```



I 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 문자를 사용할 때 대체로 아스키 문자 집합을 기본으로 사용합니다.
- 아스키 코드는 우리가 일반적으로 사용하는 각각의 문자에 0 ~ 127까지 붙여준 번호를 말합니다.
- 문자가 파일에 저장될 때는 문자에 대응되는 아스키 코드 값의 이진 데이터로 변환되어 저장됩니다.
- 문자를 어떤 규칙에 따라서 이진화(Binary: 0,1로 표현)하여 저장하는 것을 인코딩(encoding)이라고 합니다.
- 그리고 다시 문자로 바꾸는 작업을 디코딩(decoding)이라고 합니다.





```
Matt_Damon = "HOWALIVE".encode()

for idx, x in enumerate(Matt_Damon):
    print ("{:02x}".format(Matt_Damon[idx]))
```

이진 데이터(byte)로 변환

I 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 파이썬 3의 문자열은 유니코드이고, 유니코드는 세계적으로 사용하는 문자 집합을 하나로 모은 것입니다.
- 31비트로 표현되는 문자 세트이며, 온전히 한 문자를 표현하려면 4byte가 필요합니다.
- 파이썬의 문자열은 다수의 byte를 하나의 문자로 해석하는 유니코드 문자의 모임입니다.
- 참고로 '\u'를 사용하여 유니코드 문자를 지정할 수 있습니다.

```
"\ud30c\uc774\uc36c"
"\u0048\u004f\u0057\u0041\u004c\u0049\u0056\u0045"
'HOWALIVE'
```

- 유니코드는 4byte로 표현하는데 1byte, 2byte로 표현 할 수 있는 문자를 4byte로 표현을 하면 공간을 낭비 시킵니다. 또 이로 인하여 호환성 문제, 바이트 저장 순서등 문제가 발생됩니다.
- 따라서 호환성이 있으며 메모리를 절약할 수 있는 인코딩을 사용하는게 좋습니다. 이 때 사용하는 인코딩 기법인 UTF-8입니다.
- encoding을 사용하면 bytes 자료형으로 변환됩니다. 바이트는 문자열과 같이 변경 불가능 자료형이며, 문자열이 지원하는 대부분의 메소드를 지원합니다. 하지만 bytes와 문자열 간의 직접적인 연산은 허용되지 않습니다.

I 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 아래 화면처럼 byte와 문자열간의 연산은 허용되지 않습니다.

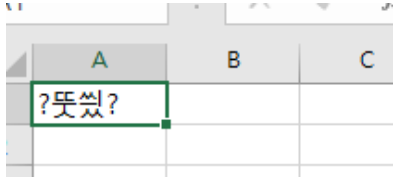
```
In [158]: Matt_Damon + "like Apple"
Traceback (most recent call last):

  File "<ipython-input-158-157caebd1433>", line 1, in <module>
    Matt_Damon + "like Apple"

TypeError: can't concat bytes to str
```

- UTF방식은 UTF-8과 UTF-16이 있습니다.(나머지 방식은 존재하지만 거의 사용하지 않습니다.)
- UTF-8은 가장 널리 쓰이는 유니코드 인코딩인데 아스키 코드는 1byte로 인코딩되고 한글의 경우 3byte로 인코딩 됩니다. (문자의 특성에 따라서 용량이 동적으로 변화하기 때문에 가변길이 인코딩이라고 하기도 합니다.)

```
"파이썬".encode('utf8')
b'\xed\x8c\x8c\xec\x9d\xb4\xec\x8d\xac'
```

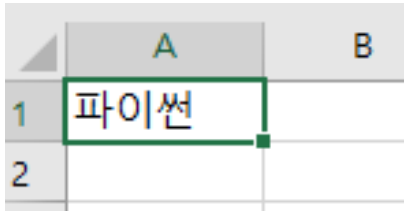
- 하지만 UTF-8로 저장된 파일은 BOM(Byte Order Mark) 정보가 없기 때문에 엑셀에서 파일을 오픈하게 되면 아래와 같이 나옵니다. (편집기에서 정상적으로 출력됩니다.)

```
f = open("gggg.csv", "w", encoding="utf8")

f.write("파이썬")
3

f.close()
```



I utf-8-sig로 저장하기

- BOM 정보가 추가된 utf-8-sig으로 인코딩하면 아래처럼 정상적으로 출력이 됩니다.



- 파일을 csv로 저장한다면 utf-8-sig를 사용하세요.
- BOM 표시인 FF FE가 추가된 것을 확인할수 있습니다.
- (참고 FF, FE은 Little Endian 방식으로 되어 있다는 의미입니다. 그냥 참고만...)
- 윈도우는 기본적으로 'cp949' 문자 집합을 사용합니다.
- - 이메일 전송하고 한글이 깨지시는 분들은 cp949로 해보시기 바랍니다.

I 파일 읽기와 쓰기

- 프로그램이 생성한 데이터를 계속 유지하고 싶다면 데이터를 보관할 방법이 필요합니다.
- 반대로 보관된 데이터를 읽어오는 기술도 필요합니다.
- 키보드 입력을 읽어 화면에 출력했던 것처럼 파일로부터 입력을 받거나 출력하는 것 역시 가능합니다.
- 형식
  - open(파일 경로, mode, encoding="")
  - 파일 경로는 파일 이름을 포함하며 절대 경로와 상대 경로 모두 사용 가능
  - open 함수의 mode

모드	설명
r	읽기 전용(기본 값)
w	쓰기 전용. 기존 내용 삭제
x	새 파일을 쓰기 전용 열기
a	기존 파일이 있는 경우 맨 뒤에 추가
b	이진 모드
t	텍스트 모드(기본값)
+	업데이트 용도로 열기

- 파일은 open() 함수를 사용하여 파일을 읽고, write() 함수를 사용하여 기록합니다.
- close() 함수를 이용하여 파일을 닫아줘야 기록이 완성됩니다.

I 절대 경로와 상대 경로 알기

- 절대 경로
  - 파일 시스템의 Root로부터 시작하는 전체 경로를 의미합니다.
  - C:\test 라고 표시하면 절대 경로입니다.

- 상대 경로
  - 현재 작업 폴더에서 시작하는 경로
  - 파일 이름도 경로에 해당합니다.
  - 현 위치에서 .\test라고 하면 상대경로입니다.

- 현재 경로 구하기 (표준 라이브러리 os 사용)
  - os.getcwd()를 사용하여 현재 경로를 구할 수 있다.
- 현재 경로 변경하기
  - os.chdir()를 사용하여 경로를 변경할 수 있다.

I 파일의 메소드 알아보기

- read() 메소드를 사용하여 전체 txt파일을 읽어옵니다.
- read() 메소드를 사용하여 한번 읽어 들인 파일은 다시 실행해도 값을 읽어 올수 없습니다.
- 그 이유는 파일의 읽은 위치가 가장 마지막으로 설정되어 있기 때문입니다.
- tell() 메소드를 사용하여 파일에서 읽은 위치를 알수가 있습니다.
- seek() 메소드를 통해서 파일의 위치를 변경할 수 있습니다.
- read() 메소드를 사용하여 읽은 파일의 위치는 마지막에 위치, 다시 read()를 호출해도 값이 나타나지 않습니다.
- seek(0)으로 하여 파일의 위치를 0번으로 이동하여 다시 read()를 사용하면 그 위치부터 읽어 올 수 있습니다.
- readline() 메소드를 사용하면 한줄씩 읽어 올 수 있습니다.
- readlines() 메소드를 사용하면 라인 단위로 전체 파일을 읽어올 수 있습니다.
- readlines()의 반환값은 list 형식으로 반환합니다.
- readlines() 메소드는 파일의 내용을 메모리에 담아서 사용하기 때문에 대용량 파일인 경우 시스템이 멈출수도 있습니다.
- with 구문을 사용하면 메모리를 올려서 사용하는 방식이 아니라 대용량 파일도 처리할 수 있습니다.

정리

- 기본 연산자
- 비교 연산자
- 논리 연산자
- 연산자 우선순위

감사합니다

FAST CAMPUS
ONLINE
서찬웅 강사.