IPA 주관 인공지능센터 기본(fundamental) 과정

- **GitHub link: here**
- **E-Mail: windkyle7@gmail.com**

File I/O

먼저, `hello.txt` 라는 텍스트 파일을 생성하였다.

In [1]:

```
%%writefile hello.txt
hello
```

```
Writing hello.txt
```

생성한 `hello.txt` 라는 **file** 객체를 `open` 함수를 통해 불러온다.

In [2]:

```
file = open('hello.txt')
```

아래의 결과를 살펴보면 읽기 모드로 불러들였는데, 기본 모드가 읽기 모드로 정해져있기 때문에 별다른 인자값을 넣어주지 않고 불러오는 데 성공하였다.

In [3]:

```
file
```

Out[3]:

```
<_io.TextIOWrapper name='hello.txt' mode='r' encoding='UTF-8'>
```

파일 입출력 뿐만 아니라 모든 **Stream** 데이터들은 메모리에 올리기 위해 `open` 을 한 후에 더 이상 사용하지 않을 때, 메모리에서 `close` 하여 해제시켜야한다.

In [4]:

```
file.close()
```

쓰기 모드는 다음과 같이 인자값에 `'w'` 를 넣어주면 사용할 수 있다.

In [5]:

```
file = open('hello.txt', 'w')
```

In [6]:

```
file.write('Hello, World!')
```

Out[6]:

```
13
```

In [7]:

```
file.close()
```

쓰기 모드가 잘 되었는지 확인해본다.

In [8]:

```
file = open('hello.txt')
```

In [9]:

```
file.read()
```

Out[9]:

```
'Hello, World!'
```

In [10]:

```
file.close()
```

In [11]:

```
%%writefile test.txt
test1
test2
test3
test4
test5
```

```
Writing test.txt
```

`read`한 후 계속 `read`를 시도하면 더 이상 읽을 데이터가 없으므로 `''`를 반환한다.

In [12]:

```
file = open('test.txt')
```

In [13]:

```
file.read()
```

Out[13]:

```
'test1\ntest2\ntest3\ntest4\ntest5\n'
```

In [14]:

```
file.read()
```

Out[14]:

```
''
```

In [15]:

```
file.read()
```

Out[15]:

```
''
```

In [16]:

```
file.close()
```

파일 객체의 `readline` 메소드로 각 줄마다 읽을 수 있다.

In [17]:

```
file = open('test.txt')
```

In [18]:

```
file.readline()
```

Out[18]:

```
'test1\n'
```

In [19]:

```
file.readline()
```

Out[19]:

'test2\n'

In [20]:

```
file.readline()
```

Out[20]:

'test3\n'

In [21]:

```
file.readline()
```

Out[21]:

'test4\n'

In [22]:

```
file.readline()
```

Out[22]:

'test5\n'

In [23]:

```
file.readline()
```

Out[23]:

''

In [24]:

```
file.readline()
```

Out[24]:

''

In [25]:

```
file.close()
```

`readlines` 메소드는 각 줄에 해당하는 데이터를 리스트로 반환한다.

In [26]:

```
file = open('test.txt')
```

In [27]:

```
file.readlines()
```

Out[27]:

```
['test1\n', 'test2\n', 'test3\n', 'test4\n', 'test5\n']
```

In [28]:

```
file.close()
```

In [29]:

```
%%writefile encoding.txt
안녕
안녕하세요
Hi
Hello
```

```
Writing encoding.txt
```

파일 인코딩 형식을 지정할 때는 인자값 `encoding`을 지정해주어야 한다.

In [30]:

```
file = open('encoding.txt', encoding='cp949')
```

In [31]:

```
file.read()
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
<ipython-input-31-f3fc120c03c1> in <module>
----> 1 file.read()

UnicodeDecodeError: 'cp949' codec can't decode byte 0xec in position 0: illegal multibyte sequence
```

In [32]:

```
file.close()
```

In [33]:

```
file = open('encoding.txt')
```

In [34]:

```
file.read()
```

Out[34]:

```
'안녕\n안녕하세요\nHi\nHello\n'
```

In [35]:

```
file.close()
```

파이썬은 기본적으로 `유니코드(UTF-8)`을 지원한다. 현재 실습 환경은 `Linux 18.04 LTS`인데, 아래의 코드를 통해 표준 입력 인코딩 형식이 `UTF-8`이라는 결과가 나왔다. 윈도우를 사용중이라면 `cp949`가 기본으로 나올 것이다.

In [36]:

```
import sys
```

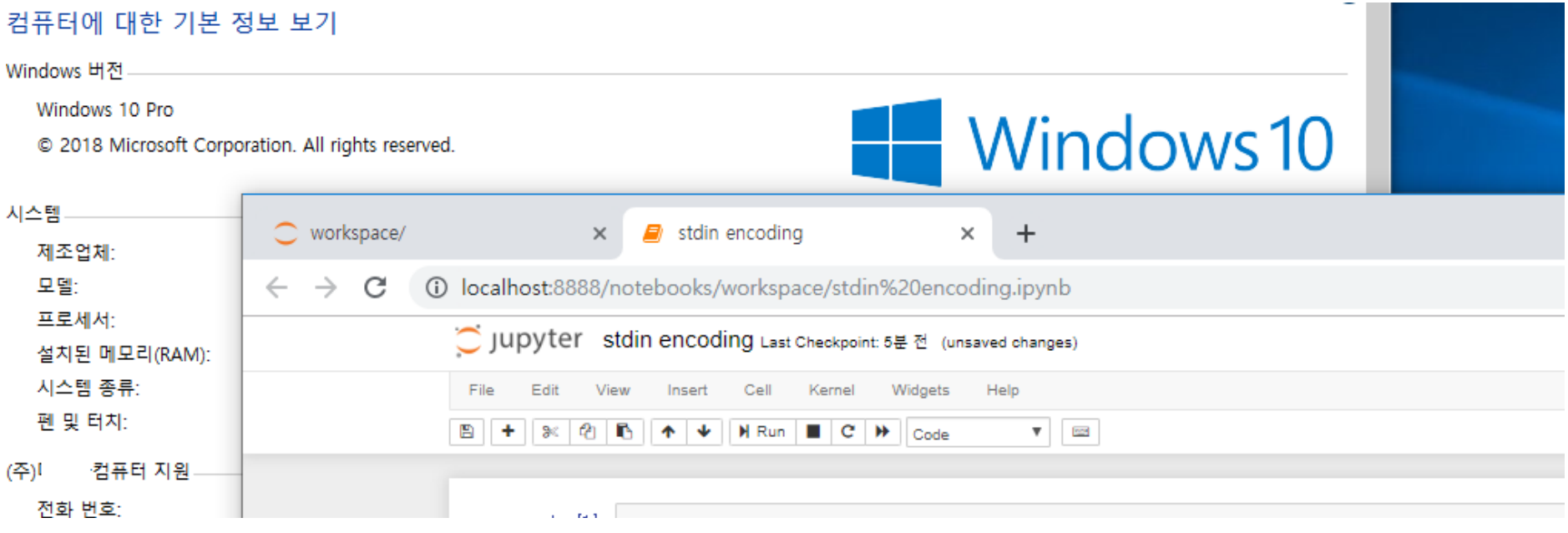
In [37]:

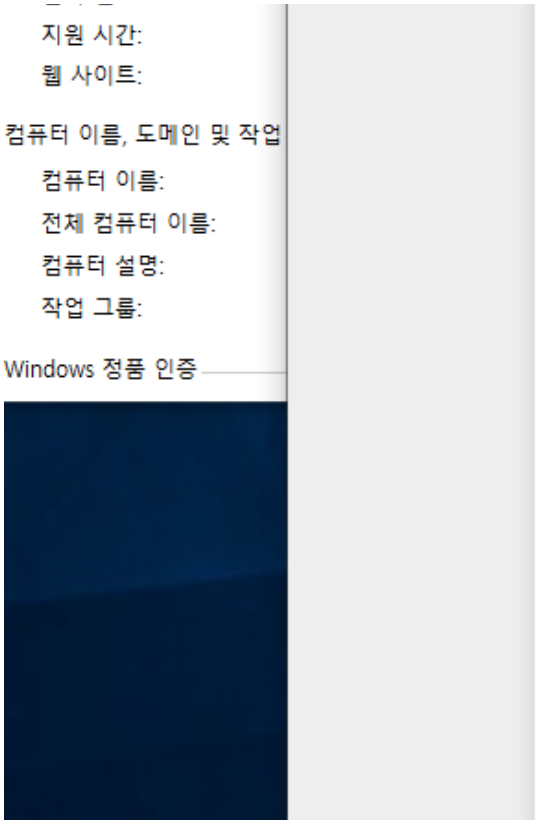
```
sys.stdin.encoding
```

Out[37]:

```
'UTF-8'
```

Windows 10 기준






```
In [1]: import sys

In [2]: sys.stdin.encoding
Out [2]: 'cp949'

In [3]: %%writefile encoding.txt
        안녕
        안녕하세요
        Hi
        Hello
        Overwriting encoding.txt

In [4]: file = open('encoding.txt')

In [5]: file.read()
        ---------------------------------------------------------------------------
        UnicodeDecodeError                        Traceback (most recent call last)
        <ipython-input-5-f3fc120c08c1> in <module>
        ----> 1 file.read()

        UnicodeDecodeError: 'cp949' codec can't decode byte 0xec in position 0: illegal multibyte sequence
```

File I/O Mode

문자	의미
'r'	읽기용으로 연다. (기본값)
'w'	쓰기용으로 연다. 파일을 먼저 자른다.
'x'	독점적인 파일 만들기용으로 연다. 이미 존재하는 경우에는 실패한다.
'a'	쓰기용으로 연다. 파일이 존재하는 경우는 파일의 끝에 덧붙인다.
'b'	바이너리 모드
't'	텍스트 모드 (기본값)
'+'	갱신(읽기 및 쓰기)용으로 디스크 파일을 연다.

파일을 매번 `open` 하였으면 항상 `close`를 해야한다. 이러한 작업을 `with` 키워드를 통해 한번에 처리할 수 있다.

with (Context Manager)

with (Context Manager)

`with` 문은 블록의 실행을 컨텍스트 관리자가 정의한 메서드들로 감싸는 데 사용된다. 흔히 `try-except-finally` 사용 패턴을 편리하게 재사용할 수 있도록 캡슐화할 수 있도록 한다.

하나의 `item`을 사용하는 `with`문의 실행은 다음과 같이 진행된다.

1. 컨텍스트 관리자를 얻기 위해 컨텍스트 표현식 (`with_item`에 주어진 `expression`) 의 값을 구한다.
2. 나중에 사용하기 위해 컨텍스트 관리자의 `__exit__()`가 로드된다.
3. 컨텍스트 관리자의 `__enter__()` 메소드를 호출한다.
4. `with` 문에 타깃이 포함되었으면, 그것에 `__enter__()`의 반환 값을 대입한다.

주석 `with` 문은 `__enter__()` 메소드가 에러 없이 돌아왔을 때, `__exit__()`가 항상 호출됨을 보장한다. 그래서 타깃에 대입하는 동안 에러가 발생하면, `with` 문 안에서 에러가 발생한 것과 같이 취급된다. (아래의 6단계를 참고.)

1. `suite`가 출력된다.
2. 컨텍스트 관리자의 `__exit__()` 메소드를 호출한다. 예외가 `with` 문 안에있는 프로시저를 종료되도록 만들었다면, 그것의 형, 값, 트레이스백이 `__exit__()` 의 인자로 전달된다. 그렇지 않으면 세 개의 `None`이 인자로 공급된다. `with` 문 안에서 예외 때문에 종료되었고, `__exit__()` 메소드의 반환 값이 `False`면, 그 예외를 다시 일으킨다. 반환 값이 `True`이면, 예외를 억누르고, `with` 문 뒤에 오는 문장으로 실행을 계속한다. `with` 문 안에서 예외 이외의 이유로 종료되면, `__exit__()`의 반환 값은 무시되고, 해당 종료의 종류에 맞는 위치에서 실행을 계속한다.

In [38]:

```
class A:
    def __enter__(self):
        print('Call A __enter__()')

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('Call A __exit__()')
```

In [39]:

```
class B:
    def __enter__(self):
        print('Call B __enter__()')

    def __exit__(self, exc_type, exc_val, exc_tb):
        print('Call B __exit__()')
```

In [40]:

```
with A() as a:
    with B() as b:
        print('suite')
```

```
Call A __enter__()
```

```
Call A __enter__()
Call B __enter__()
suite
Call B __exit__()
Call A __exit__()
```

아래처럼 한줄에 축약하여 쓸 수 있다.

In [41]:

```
with A() as a, B() as b:
    print('suite')
```

```
Call A __enter__()
Call B __enter__()
suite
Call B __exit__()
Call A __exit__()
```

아래의 예제에서 `with` 문 안에서 예외가 발생하면 예외가 발생한 부분을 실행하지 않고 바로 `__exit__()`를 호출하는 것을 확인할 수 있다.

In [42]:

```
with A() as a:
    suite
    print('suite')
```

```
Call A __enter__()
Call A __exit__()

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-42-f7db34e292c6> in <module>
      1 with A() as a:
----> 2     suite
      3     print('suite')

NameError: name 'suite' is not defined
```

`with` 문을 사용하면 `__enter__()` 메소드와 `__exit__()` 메소드를 알아서 수행하기 때문에 파일 **IO**를 편리하게 할 수 있다.

In [43]:

```
with open('test.txt') as file:
    print(file.read())
```

```
test1
test2
test3
```

```
test4
test5
```