Lecture 9

문자열과 입출력

문자열(str)

- 이전에 배운 문자열(string) 자료형에 대해 복기해보자
- 문자열 자료형은 **str**로 표기되며 **문자들의 시퀀스**라고 생각하면 된다
- 문자열은 **작은 따옴표(') 혹은 큰 따옴표(")의 쌍**으로 문자를 둘러 싸야 정상적으로 인식된다
  - 작은 따옴표와 큰 따옴표의 조합으로는 문자열을 둘러싸면 안 됨 (문자열이 덜 끝난 것으로 인식)
  - Curly quote(‘ 혹은 “)가 아닌 **Straight quote(' 혹은 ")**여야 인식이 됨 (일반적인 IDE에서는 따옴표를 키 스트로크하면 자동적으로 Straight quote로 작성됨)
- 여러 줄의 문자열은 **연속된 3개의 작은 따옴표(''') 혹은 큰 따옴표들(""")의 쌍**으로 문자를 둘러 싼다

예제1

```
1  one = 1; one_str = '1'; one_str2 = "1"
2  one_str3 = '''1'''
3  one_str4 = """1"""
4  print(one, type(one))
5  print(one_str, type(one_str))
6  print(one_str2, one_str3, one_str4) # print 문에
   여러개의 문자열을 넣으면 자동으로 end=" "가 적용
```

→

```
1 <class 'int'>
1 <class 'str'>
1 1 1
```

잠깐! *한 줄에 여러 줄의 코드를 작성하고 싶으면 줄의 구분을 위해 세미콜론(;)을 기입하면 된다*

문자열(str)

- 문자열 객체의 비교로는 '값' 비교와 '주소' 비교가 있다
- 값 비교는 비교 연산자 중에 하나인 **==**를 사용하고 주소 비교는 **is**를 활용한다

예제2

```
1  one = 1; one_str = '1'; one_str2 = "1"
2  one_str3 = '"1"'; one_str4 = """1"""
3  print(one == one_str) # 같은 "값"을 지니는지 확인
4  print(one_str == one_str2 == one_str3 == one_str4)
5  print(one_str is one_str2) # 같은 "주소"를 가지는지 확인
6  one_str_new = str(one)
7  print(one_str_new == one_str)
8  print(one_str_new is one_str)
9  print(id(one)) # 주소 확인 방법
10 print(id(one_str), id(one_str2), id(one_str3), id(one_str4))
11 print(id(one_str_new)) # 다른 주소를 지님을 확인할 수 있음
```

→

```
False
True
True
True
False
4383254832
4385793776 4385793776 4385793776
4385793776
4436009712
```

문자열 출력

- 다른 프로그래밍 언어에서는 문자열(str; string)과 하나의 문자(char; character)를 서로 자료형적으로 구분하는 경우가 있다: 문자열은 **큰 따옴표(")**, 문자는 **작은 따옴표(')**
- 파이썬에서는 무엇을 써야 한다는 정답은 없지만 일반적으로 위와 같은 관례를 따른다

예제3

```
1  print('a', type('a'))
2  print("abc", type("abc"))
```

→

```
a <class 'str'>
abc <class 'str'>
```

- 하나의 print 함수 안에 여러개의 문자열을 작성하는 방법은 다음과 같다:

예제4

```
1  print("Hi", "There") # 콤마는 문자열 간에 띄어쓰기 자동 적용
2  print("Hi"+"There") # 덧셈 연산자는 문자열을 concatenate(연결)시킴
3  print("Hi""There") # 굳이 덧셈 연산자를 안 붙여도 concat됨; but 가독성 bad
```

→

```
Hi There
HiThere
HiThere
```

문자열 출력

- 이전 시간에 배운 문자열의 특수 문자에 대해 다시 짚고 넘어가자
- 이들의 정식 명칭은 **이스케이프 문자(escape character)**이다
- 이스케이프 문자는 문자열 안에서 항상 **백슬래시 \ 를 접두로 하여 작성**된다

예제5

```
1  print("New line\n")
2  print("Backslash\\")
3  print("Double quote\"")
4  print("Back space\b")
5  print("Horizontal tab |\t|")
6  print("Carriage return\rWow")
```

```
New line

Backslash\
Double quote"
Back spac
Horizontal tab |       |
Wowriage return
```

문자열 출력

- 문자열 출력 포맷팅에 대해서도 소개한다
- 문자열 안에 특정 (변수의) 값을 함께 출력하고 싶을 때 사용하는 방식이며 다음과 같은 방법이 존재한다
  - **% 연산자**
  - format 옵션
  - 포맷 문자열 리터럴 (f-string)
- %d: int, %f: float, %s: str, %c (문자 한개), %o (8진수), %x (16진수), %% (% 그 자체)
- “%0nd” % (숫자): 숫자로부터 n자리수만큼의 정수는 최소 만들어야 되고, 부족하면 0을 패딩한다

예제6

```
1  a = "%03d" % (7); b = "%03d" % (77)
2  c = "%03d" % (777); d = "%03d" % (7777)
3  print(a, b, c, d)
```

→

```
007 077 777 7777
```

- “%.nf” % (숫자): 숫자로부터 실수 를 출력하며 소수점 아래로 n자리 만큼 출력한다

예제7

```
1  print("%.2f"%3.3333)
2  print("%f"%3.3333) # 소수점 여섯자리 출력이 디폴트
3  print("%(second)f, %(first).2f"%{"first":3.3, "second": 2.2})
```

→

```
3.33
3.333300
2.200000, 3.30
```

문자열 출력

- 문자열 출력 포맷팅에 대해서도 소개한다
- 문자열 안에 특정 (변수의) 값을 함께 출력하고 싶을 때 사용하는 방식이며 다음과 같은 방법이 존재한다
  - % 연산자
  - **format 옵션**
  - 포맷 문자열 리터럴 (f-string)
- 파이썬 표현식 값을 바인딩할 부분을 **중괄호 {}** 로 표시하고 **.format()** 안에 바인딩할 값을 기술한다

예제8

```
1  seven = 7
2  print("Seven is {}".format(seven))
3  print("{0:.2f}".format(seven)) # format() 안의 0번째 값
4  print("{number:.2f}".format(number = seven)) # number 변수 값
```

→

```
Seven is 7
7.00
7.00
```

예제9

```
1  Score_list = [["Nubzuki", 100]]
2  print("{name} got a score of {score}.".format(name=Score_list[0][0], score = Score_list[0][1]))
```

→

```
Nubzuki got a score of 100.
```

문자열 출력

- 문자열 출력 포맷팅에 대해서도 소개한다
- 문자열 안에 특정 (변수의) 값을 함께 출력하고 싶을 때 사용하는 방식이며 다음과 같은 방법이 존재한다
  - % 연산자
  - format 옵션
  - **포맷 문자열 리터럴 (f-string)**
- 파이썬 3.6버전부터 지원하는 방식으로 문자열에 f 나 F를 접두하고 표현식을 **중괄호 {}** 안에 작성한다

예제10

```
1  pi = 3.141592
2  print(f"파이는 {pi}")
3  print(f"파이는{pi: .2f}")
```



```
파이는 3.141592
파이는 3.14
```

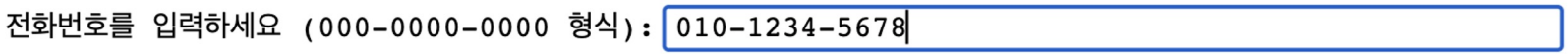
- f-string 포맷팅은 가장 직관적이며 표현이 간략하여 세 방식 중 가장 추천하는 방식이다

문자열 입력

- 문자열 입력(콘솔 입력)으로는 **input 함수**를 사용한다
- 괄호 안 파라미터에 문자열을 넣으면 입력창 직전에 안내글로써 함께 출력할 수 있다

예제11

```
1  number = input("전화번호를 입력하세요 (000-0000-0000 형식):")
2  number_list = number.split('-')
3  print(number_list)
```




```
전화번호를 입력하세요 (000-0000-0000 형식):010-1234-
5678
['010', '1234', '5678']
```

잠깐! *콘솔이란 컴퓨터 프로그래밍에서 디버깅, 테스트, 로깅 등에 사용되는 인터페이스이다 (명령 프롬프트 창)*

파일 입출력

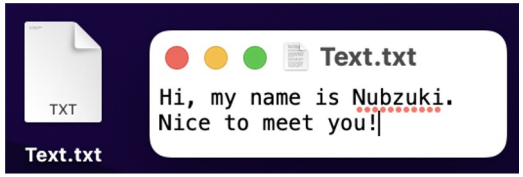
- 파이썬에서 파일을 열람할때에 사용하는 함수는 **open 함수**이다 (기본 내장함수)

```
def open(file, mode='r', buffering=None, encoding=None, errors=None, newline=None, closefd=True):
```

- 보통은 file과 mode 인자만을 입력하며 file은 꼭 입력해야 할 파라미터이며 mode는 매우 권고한다
- file에는 열람할 파일 경로를 입력한다
- mode에는 파일을 열람하고 다루는 방식을 받는다; 정규표현식: [rxwa](\+)?(tb)?
  - 'r': 읽기모드(read); 디폴트 값; 파일을 읽기 위한 옵션; 파일이 존재하지 않으면 에러가 뜬다
  - ‘x’: 쓰기모드; 파일을 새롭게 생성 후 내용을 작성; 파일이 존재하면 에러가 뜬다
  - 'w': 쓰기모드(write); 파일에 덮어 쓰기 위한 옵션 (이미 파일이 존재하면 **모든 내용을 리셋** 후 내용을 작성하며 파일이 존재하지 않으면 파일을 생성한 뒤 내용을 작성)
  - 'a': 추가모드(append); w와 비슷하게 쓰기모드이지만 파일이 존재하면 **커서를 맨 뒤**에다가 가져다가 내용을 이어서 작성하게 한다 (파일이 존재하지 않으면 파일을 생성한 뒤 내용을 작성)
  - '+': 읽기쓰기모드 지원
  - 't': 텍스트모드; 디폴트 값 / 'b': 바이너리모드 (바이트단위 데이터 기록에 사용)

파일 입출력

- 예제 코드들을 통해 파일 입출력과 익숙해지자
- 스크립트 파일과 동일한 경로에 다음과 같이 Text.txt를 생성하자



예제12

```
1  f = open("./Text.txt", 'r') # 읽기모드
2  line = f.readline()
3  print(line)
4  line2 = f.readline()
5  print(line2)
6  f.close() # 파일을 열은 뒤에는 꼭 닫아주자
```

→

```
Hi, my name is Nubzuki.

Nice to meet you!
```

예제13

```
1  f = open("./Text.txt", 'w') # 쓰기모드
2  f.write("안녕하세요\n처음뵙겠습니다")
3  f.close()
4  f = open("./Text.txt", 'r') # 읽기모드
5  print(f.readlines()) # 파일의 모든 줄 읽어서 리스트로 보관
6  f.close()
```

→

```
['안녕하세요\n', '처음뵙겠습니다']
```

파일 입출력

- readlines에서 생기는 줄바꿈 이스케이프 문자 \n 를 제거하려면 **strip** 혹은 **rstrip** 함수를 사용하자

예제14

```
1  f = open("./Text.txt", 'w')
2  f.write("안녕하세요\n처음뵙겠습니다")
3  f.close()
4  f = open("./Text.txt", 'r')
5  lines = f.readlines()
6  for index, line in enumerate(lines):
7          lines[index] = line.strip() # 문자열 양끝 공백들을 제거
8  print(lines)
9  f.close()
```

→ ['안녕하세요', '처음뵙겠습니다']

파일 입출력

- x 모드는 안전하게 기존에 없던 파일을 새로 생성한 후 내용을 쓰게 해준다

예제15

```
1  f = open("./Text2.txt", 'x')
2  f.write("새로운 파일 생성")
3  f.close()
4  f = open("./Text2.txt", 'r')
5  lines = f.readlines()
6  print(lines)
7  f.close()
```



```
['새로운 파일 생성']
```

- 파일경로가 겹치면 에러가 발생한다

예제16

```
1  f = open("./Text.txt", 'x')
2  f.write("새로운 파일 생성")
3  f.close()
```



```
FileExistsError: [Errno 17] File
exists: './Text.txt'
```

파일 입출력

- a 모드는 기존에 파일이 있으면 뒤에 이어서 내용을 추가, 없으면 파일을 생성 후 내용을 쓰게 해준다

예제17

```
1  f = open("./Text2.txt", 'a')
2  f.write("!!!")
3  f.close()
4  f = open("./Text2.txt", 'r')
5  lines = f.readlines()
6  print(lines)
7  f.close()
```

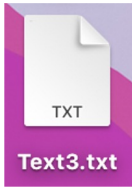
→

```
['새로운 파일 생성!!!']
```

예제18

```
1  f = open("./Text3.txt", 'a')
2  f.write("!!!")
3  f.close()
4  f = open("./Text3.txt", 'r')
5  lines = f.readlines()
6  print(lines)
7  f.close()
```

→




```
['!!!']
```

파일 입출력

- a+ 모드는 기존에 파일이 있으면 뒤에 이어서 내용을 추가, 없으면 파일을 생성 후 내용을 쓰게 해준다
  또한 읽기와 쓰기 모드를 둘다 지원한다

예제19

```
1  with open("./Text3.txt", 'a+') as file: # 파일 열람 방식의 또 다른 형태 (with open as)
2          file.write("???")
3          file.seek(0) # 파일 포인터의 맨 처음으로의 이동 (seek 함수)
4          lines = file.read() # 내용 전체를 하나의 문자열로 받기
5          print(lines)
```

→ !!!???

예제20

```
1  with open("./Text3.txt", 'a+t') as file: # 디폴트로 마지막에 t가 붙여져왔음
2          file.write("!!!")
3          file.seek(0)
4          lines = file.read()
5          print(lines)
```

→ !!!???!!!

파일 입출력

- b 모드는 바이너리 모드로 바이트 단위의 데이터를 다룰 때 사용한다

예제21

```
1  file=open("files.txt","wb") # 바이너리 쓰기모드
2  numbers=[5, 10, 15, 20]
3  array=bytearray(numbers)
4  print(array)
5  file.write(array)
6  file.close()
7  file=open("files.txt","rb") # 바이너리 읽기모드
8  array = bytes(file.read())
9  print(array)
10 print(list(array))
11 file.close()
```

```
bytearray(b'\x05\n\x0f\x14')
b'\x05\n\x0f\x14'
[5, 10, 15, 20]
```

잠깐! *bytes와 bytearray 자료형은 1 byte 단위의 값을 연속적으로 저장하는 시퀀스 자료형.*
bytes는 원소 변경이 안되지만 bytearray는 가능하다