Lecture #7 | 문자열 처리
파일 입출력

지난 시간에 다룬 내용

- 사전(dict)
  - 사전의 정의와 사용

이번 시간에 다룰 내용

- 문자열 함수
  - 문자열 서식화
  - `split()`, `splitlines()`
- 파일입출력

문자열 함수 (일부)

- 문자열 함수의 용법: *string.function(arguments)*
  - 나누기/붙이기: **split()**, join(), **splitlines()**
  - 문자열 검색/치환: find(), rfind(), count(), replace()
  - 포매팅: rjust(), ljust(), center(), **format()**
- string에 위치할 수 있는 것들
  - 문자열
  - 문자열을 포함한 변수
  - 문자열값을 반환하는 함수
- 일반적으로 문자열 함수는 문자열을 반환값으로 가짐
  - 예외: split(), splitlines()는 리스트를 반환함

문자열 서식화 (string formatting)

- python에서 문자열 서식화
  - 서식문자열에 따라, (변수에 저장된) 값의 자리수, 표현방식을 서식화함
  - 서식화된 '**문자열**'을 반환 → 변수에 대입하거나, `print()` 함수를 이용하여 출력할 수 있음
    - 참고: C언어의 `printf()` 함수가 아닌, `sprintf()` 함수에 해당
- python에서 지원하는 문자열 서식화 방식
  - **Printf-style string formatting** (all python versions)
    - C언어의 `printf()` 함수와 유사학 서식문자열을 지원
  - **`format()` function with format string syntax** (python 3, python 2.6+)
    - 서식문자열의 표현이 더 풍부해짐
  - **Literal string interpolation** (python 3.6+)
    - 서식문자열에 수식 혹은 변수의 이름을 적을 수 있는 기능을 추가
  - **Template string** (python 2+): printf-style과 유사

printf-style string formatting (% operator*)

- 사용형태: *format* `%` *values*
  - *format*: 서식 문자열로, 다음 형태의 변환명시자(conversion specifier)를 포함
    - %로 시작하고, 변환형을 나타내는 하나의 문자로 끝남
    - mapping key, conversion flag, 최소 문자 폭, 정확도를 명시할 수 있음
  - *values*: 다음 중 하나의 형식
    - 한 개의 값 (서식 문자열이 하나의 값을 요구할 때)
    - 서식문자열이 요구하는 아이템 수와 값은 수의 값이 있는 튜플
    - 한 개의 사전(dict)과 같은 매핑 객체 (mapping object)
- 기능: 서식 문자열에 따라 *values*에 주어진 값들을 원하는 형식의 **'문자열'**로 변환함
- 단점: 튜플, 사전, 객체 등 복잡한 자료형의 값을 출력할 때 제한점이 많음
- 참고: C언어의 `printf()` 혹은 `sprintf()` 함수에서 서식문자열과 유사함

* string formatting or interpolation operator라고도 불림

변환 명시자 (conversion specifier) 형식

1. %로 시작
2. (Optional) mapping key
   - ()안에 문자들로 주어짐
   - 맵핑 객체 형태의 *values*에서 키가 ()안의 문자들인 값으로 대체됨
3. (Optional) conversion flag (일부)
   - `'0'`: 수의 경우, 남는 공간이 `0`으로 채워짐
   - `'-'`: 변환된 문자열이 왼쪽에 정렬됨
4. (Optional) 최소 문자 폭: 출력되는 값의 최소 문자의 수를 지정하는 정수
5. (Optional) 정확도: `'.'`와 숫자로, 소수 부분의 자리수을 명시
6. (Optional) length modifier: python에서는 사용되지 않음
7. 변환형을 나타내는 하나의 문자로 끝남 (다음 페이지 참고)

Reference: 변환형

Conversion	Meaning
`'d' or 'i'`	Signed integer decimal.
`'o'`	Signed octal value.
`'u'`	Obsolete type – it is identical to 'd'.
`'x' or 'X'`	Signed hexadecimal (lowercase (`'x'`) or uppercase(`'X'`).
`'e' or 'E'`	Floating point exponential format (lowercase (`'e'`) or uppercase(`'E'`).
`'f' or 'F'`	Floating point decimal format.
`'g' or 'G'`	Floating point format. Uses lowercase (`'g'`) or uppercase (`'G'`) exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
`'c'`	Single character (accepts integer or single character string).
`'r', 's', 'a'`	String (converts any Python object using repr(), str(), and ascii(), respectively)
`'%'`	No argument is converted, results in a '%' character in the result.

예시: printf-style string formatting

```
var = 42
string = 'Answer'
pi = 3.141592653589793
s1 = 'The %s is %d.' % (string, var)
print(s1)
print('0123456789' * 2)
print('%10d' % var)
print('%010d' % var)
print('%-10d' % var)
print('%f' % pi)
print('%.2f' % pi)
print('%.10f' % pi)
print('%10.2f' % pi)
print('%g, %g, %g' % (var, pi, 10000000000))
print('%(name)s: %(average)g' %
        {'name': 'Alice', 'average': 99.9})
```

```
The Answer is 42.
01234567890123456789
________42
0000000042
42________
3.141593
3.14
3.1415926536
______3.14
42, 3.14159, 1e+10
Alice: 99.9
```

* 빈칸은 ␣로 표시함

format() 함수를 이용한 문자열 서식화

- 사용형태: *string.format(arguments)*
  - *string*: 서식 문자열
  - *arguments*: 서식 문자열에 의해 변환된 인자들
- 서식 문자열에 '{*fieldname*:*format_spec*}'의 형태가 주어지면 인자의 값의 서식을 변경함
  - *fieldname*: 어떤 인자의 값을 서식화할 지를 지정
  - *format_spec*: 어떠한 형태로 서식화할 지를 지정
- 참고: 서식화를 지정하는 서식 명시자(formatting specifier)는 printf 형태의 변환 명시자와 유사하나, 그 기능이 확장되었고 사용이 조금 더 편리하도록 변경함

예시: format() 함수

```
d = 42
f = 3.14
s = 'apple'
print('{} {} {}'.format(d, f, s))
print('{2} {0} {1}'.format(d, f, s))
print('{0} is {0}'.format(s))
print('{d} {f} {s}'.format(d=6, f=1.618, s='pineapple'))
print('{0:d} {0:f} {0:g}'.format(42))
```

```
42 3.14 apple
apple 42 3.14
apple is apple
6 1.618 pineapple
42 42.000000 42
```

예시: format() 함수 (cont.)

```
print('0123456789' * 2)
print('{:10d}'.format(42))
print('{:>10d}'.format(42))
print('{:<10d}'.format(42))
print('{:^10d}'.format(42))
print('{:10.2f}'.format(3.1415926535))
print('{:1d}'.format(42))
print('{:5.5f}'.format(3.1415926535))
```

```
01234567890123456789
________42
________42
42________
____42____
______3.14
42
3.14159
```

* 빈칸은 _로 표시함

Format string literal을 이용한 문자열 서식화

- f-string
  - 문자열 앞에 `f`를 이용하여 표시*
  - 서식 표현은 `format()` 함수와 유사
  - `'{}'` 안에 계산이 가능한 표현(expression)이 들어감
  - f-string이 실행될 때, `'{}'` 안의 표현이 연산(evaluate)됨

* python에서 문자열 앞에 한 글자를 추가하여, 특수한 문자열을 나타냄

예시: Literal string interpolation

```
pi = 3.14159265
width = 10
precision = 2
print('0123456789' * 3)
print(f'{pi}')
print(f'{pi:10.2f}')
print(f'{pi:{width}.{precision}f}')
print(f'{max(width, precision)}')
t = [42, 1024, 23]
for i in range(len(t)):
    print(f'{i}: {t[i]}')
```

```
012345678901234567890123456789
3.14159265
␣␣␣␣␣␣3.14
␣␣␣␣␣␣3.14
10
0:␣42
1:␣1024
2:␣23
```

* 빈칸은 ␣로 표시함

split(): 구분자를 기준으로 분리된 문자열의 리스트를 반환

str.split(*sep*=None, *maxsplit*=-1)

- Return a list of the words in the string, using *sep* as the delimiter string.
- If *maxsplit* is given, at most *maxsplit* splits are done (thus, the list will have at most maxsplit+1 elements).
- If *maxsplit* is not specified or -1, then there is no limit on the number of splits (all possible splits are made).
- If *sep* is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')` returns `['1', '', '2']`).
- The *sep* argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>')` returns `['1', '2','3']`).
- Splitting an empty string with a specified separator returns `['']`.

`splitlines()`: 개행문자로 분리된 문자열의 리스트를 반환

`str.splitlines([keepends])`

- Return a list of the lines in the string, breaking at line boundaries.
- Line breaks are not included in the resulting list unless *`keepends`* is given and true.
- This method splits on the following line boundaries.
- In particular, the boundaries are a superset of universal newlines.
- 참고: `split('\n')`과 유사하나, 마지막 빈 줄을 남기지 않음

예시: split(), splitlines() 함수

```
quotes = '''First, solve the problem. Then, write the code. - John Johnson
Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley
Computers are good at following instructions, but not at reading your mind. - Donald Knuth
Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods'''
list_quote = quotes.splitlines()
print(list_quote)
for quote in list_quote:
    sentence, author = quote.split(' - ')
    print('"' + sentence + '" by ' + author)
    # or print(f'"{sentence}" by {author}')
```

예시: split(), splitlines() 함수 (실행결과)

```
['First, solve the problem. Then, write the code. - John Johnson', 'Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley', 'Computers are good at following instructions, but not at reading your mind. - Donald Knuth', 'Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods']
"First, solve the problem. Then, write the code." by John Johnson
"Without requirements or design, programming is the art of adding bugs to an empty text file." by Louis Srygley
"Computers are good at following instructions, but not at reading your mind." by Donald Knuth
"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live." by John Woods
```

파일

- 파일
  - 문자, 숫자 등으로 이루어진 정보의 집합체
  - HDD, SSD, USB drive, 클라우드 등의 저장장치에 저장됨
  - 파일이 저장되어 있는 공간(파일 경로 혹은 path)과 이름(파일명)으로 구분됨
- 텍스트 파일: 여러 줄의 사람들이 인지할 수 있는 문자들로 이루어진 파일
  - 문자인코딩(예: ASCII, UTF-8, CP-949)에 따라 표현할 수 있는 문자가 달라짐
  - 참고: 파이썬 소스 파일(.py), html 파일 (.html) 등도 텍스트 파일의 일종
- 바이너리 파일: 텍스트 파일이 아닌 파일
  - 예: 아래아한글 문서파일 (.hwp), 워드파일 (.doc), 이미지파일 (.jpg, .png) 등

텍스트 파일입출력

- 파일 입출력 전후로 파일을 열고, 닫는 단계가 필요함 (컴퓨터로 문서를 편집하기 위해서 파일을 여는 것과 유사함)
  - 파일 열기 (open)
  - 파일 읽기 혹은 쓰기 (read or write)
  - 파일 닫기 (close): `with`를 사용한 경우 생략
- python에서 제일 간단한 텍스트 파일 입출력 방법
  - 파일입력
    1. `read()` 함수 이용 → 파일 전체를 하나의 문자열로 읽음
    2. `split()`, `splitlines()` 함수 이용 → 한 줄씩 처리
  - 파일출력: `print()` 함수 이용 → 화면에 인쇄하는 것처럼 파일 출력 가능

텍스트 파일 읽기

- 코드 설명
  - example.txt라는 이름의 파일의 내용을 변수 file_contents에 저장
- 일반적인 방법

```
fileobj = open('example.txt', 'rt')
file_content = fileobj.read()
fileobj.close()
```

- with를 사용 (close()를 호출할 필요가 없음)

```
with open('example.txt', 'rt') as fileobj:
    file_content = fileobj.read()
```

텍스트 파일 쓰기

- 코드 설명
  - example.txt라는 이름의 파일에 문자열 something을 쓰기
- 일반적인 방법

```
fileobj = open('example.txt', 'wt')
print('something', file=fileobj)
fileobj.close()
```

- with를 사용 (close()를 호출할 필요가 없음)

```
with open('example.txt', 'wt') as fileobj:
    print('something', file=fileobj)
```

예시: 파일입출력 함수

```
quotes = '''First, solve the problem. Then, write the code. - John Johnson
Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley
Computers are good at following instructions, but not at reading your mind. - Donald Knuth
Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods'''
with open('quotes.txt', 'wt') as f:
    print(quotes, file=f)
with open('quotes.txt', 'rt') as f:
    file_contents = f.read()
print(file_contents)
```

예시: 파일입출력 함수 (실행 결과)

```
First, solve the problem. Then, write the code. - John Johnson
Without requirements or design, programming is the art of adding bugs to an empty text file. - Louis Srygley
Computers are good at following instructions, but not at reading your mind. - Donald Knuth
Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. - John Woods
```

읽을 거리

- python 문자열 자료형과 함수: https://wikidocs.net/13
- python 문자열 서식화
  - PyFormat Using % and .format() for great good!: https://pyformat.info/
  - The 4 Major Ways to Do String Formatting in Python: https://dbader.org/blog/python-string-formatting
- References
  - printf-style string formatting: https://docs.python.org/3/library/stdtypes.html?highlight=printf#old-string-formatting
  - Format string syntax: https://docs.python.org/3/library/string.html#formatstrings
  - Literal string interpolation: https://www.python.org/dev/peps/pep-0498/

ANY QUESTIONS?