2020 20173855 지병강 파이썬 포트폴리오

이름	지병강	학번	20173855
주제	파이썬 포트폴리오 (학습 요약)		

학습 자료 & 참고 자료

1. 파이썬으로 배우는 누구나 코딩 (홍릉 과학출판사, 강환수, 신용현 지음)

2. do it! 점프 투 파이썬 (이지스퍼블리싱, 박응용 지음)

3. 인터넷

목차

1장
파이썬에 관하여

1-1. 파이썬 이란?
1-2. 파이썬의 특징
1-3. 파이썬으로 할 수 있는 것
1-4. 파이썬 설치
1-5. 파이썬과 에디터

2장
파이썬의 기초, 자료형

2-1. 숫자형
2-2. 문자열 자료형
2-3. 리스트 자료형
2-4. 튜플 자료형
2-5. 딕셔너리 자료형
2-6. 집합 자료형
2-7. 불 자료형
2-8. 변수

3장
파이썬의 구조, 제어문

3-1. if문
3-2. while문
3-3. for문

4장
함수와 프로그램의 입력과 출력

4-1. 함수
4-2. 사용자 입력과 출력
4-3. 파일 읽고 쓰기

5장
파이썬 심화 과정

5-1. 클래스
5-2. 모듈
5-3. 내장 함수

6장 연습 문제

1장 파이썬에 관하여

1-1 파이썬이란?

'파이썬'이라는 것을 어떤것일까? 필자는 항상 프로그래밍에 관련된 것들을 학습하기 이전에 내가 학습하려는 것은 무엇이고 어떤 것을 배우게 될지 에대해 미리 알고 시작하는걸 선호한다.
일반적으로 책에서 설명하는 파이썬은 다음과 같다. **'파이썬(Python)은 1990년 암스테르담의 귀도 반 로섬이 개발한 인터프리터 언어이다.'**라고 많은 책이나 인터넷에서 거의 비슷하게 설명해준다.
그렇다면 하나하나 알아보자 우선 파이썬의 사전적 의미는 고대 신화에 나오는 파르나소스 산의 동굴에 살던 큰 뱀을 뜻하며, 아폴로 신이 델파이에서 파이썬을 퇴치했다는 이야기가 전해지고 있다.
대부분의 파이썬 책 표지와 아이콘이 뱀 모양으로 그려져 있는 이유가 여기에 있다.

그렇다면 또 다른 단어에 주목하자 '인터프리터 언어' 이 단어는 무엇을 의미하는 것일까?
'인터프리터 언어'란 한 줄씨 소스 코드를 해석해서 그때그때 실행해 결과를 바로 확인할 수 있는 언어이다.

파이썬은 컴퓨터 프로그래밍 교육을 위해 많이 사용하지만, 기업의 실무

를 위해서도 많이 사용하는 언어이다. 현재 파이썬을 이용하는 기업들을 적자면 크게 다음 세가지 기업들이 있다.
구글(Google), 인스타그램(Instagram), 파일 동기화 서비스 드롭박스(Dropbox) 어떤가 많이 들어본 이름들이지 않은가?
또한, 파이썬 프로그램은 공동 작업과 유지 보수가 매우 쉽고 편하다. 그래서인지 현재 파이썬을 사용해 프로그램을 개발하는 업체들 또한 늘어가고 있는 추세다.

Worldwide, Mar 2020 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.09 %	+3.9 %
2		Java	18.84 %	-1.7 %
3		Javascript	8.1 %	-0.1 %
4		C#	7.27 %	-0.0 %
5		PHP	6.08 %	-1.1 %
6		C/C++	5.86 %	-0.2 %
7		R	3.73 %	-0.2 %
8		Objective-C	2.42 %	-0.5 %
9		Swift	2.28 %	-0.1 %
10		Matlab	1.89 %	-0.1 %

그림 2. 2020년 3월 PYPL 프로그래밍 언어 순위 (출처: http://pypl.github.io/PYPL.html).

1-2 파이썬의 특징

대략적으로 파이썬이 어떻게 만들어졌는지 알아봤다면 다음은 파이썬의 특징을 나열해보겠다.

· **파이썬은 인간다운 언어이다.**

인터프리터 언어를 사용해서 인지 처음 배울때는 대화를 하는 느낌을 받기도 했다.

· **파이썬은 문법이 쉬워 빠르게 배울 수 있다.**

파이썬을 추천한다면 아무 의심할 여지없이 이 부분 때문에 추천을 할 것이다. 실제로 지금까지 배운 프로그래밍 언어중에서 파이썬 만큼 빠르고 재밌게 배운 언어는 없었다. 그만큼 문법이 쉽고 빠르게 배울 수 있다는 장점은 파이썬에서 커다란 장점으로 평가한다.

· **파이썬은 무료지만 강력하다.**

오픈 소스인 파이썬은 무료다. 파이썬에서 제공하는 다양한 라이브러리도 사용가능하다.

· **파이썬은 간결하다.**

들여쓰기를 주의하며 코드를 작성해야 하긴 하지만 그 이유 때문에 더욱 간결하고 가독성 높은 코딩을 할 수 있다.
들여쓰기가 습관이되 여기서도 한 것이 보이는가?

이외에 몇가지가 있지만 필자는 와 닿는 부분만 기술했다.

1-3 파이썬으로 할 수 있는 것

정말 어느 책에서라도 기술했으면 좋겠는 부분이 여기부분이다. 너무나 중요하고 쉽지만 많은 책들이 간과해서 넘기는 부분이라 생각한다.
파이썬으로 할 수 있는 것들은 다음과 같이 표로 작성해보았다.

파이썬으로 할 수 있는 일들	내용
시스템 유틸리티 제작	**유틸리티란 컴퓨터 사용에 도움을 주는 여러 소프트웨어를 말한다. 파이썬은 운영체제(윈도우, 리눅스)를 바탕으로 시스템 유틸리**

	티를 만드는데 유리하다.
GUI 프로그래밍	GUI 프로그래밍이란 쉽게 말해 화면에 또 다른 윈도우 창을 만들고 그 창에 프로그램을 동작시킬 수 있는 메뉴나 버튼, 그림 등을 추가하는 것이다.
C/C++ 와의 결합	파이썬은 접착 언어라고도 불리는데, 다른 언어와 잘 어울려 결합해서 사용할 수 있기 때문이다. C와C++로 만든 프로그램을 파이썬에서 사용가능하다.
웹 프로그래밍	게시판이나 방명록에 글을 남겨 본적 있다면 이해하기 쉽다. 그러한 기능들을 바로 웹 프로그램이라 한다.
수치 연산 프로그래밍	파이썬은 NumPy라는 수치 연산 모듈을 제공한다.
데이터베이스 프로그래밍	피클이라는 모듈을 제공하고, 파이썬은 사이스베이스, 인포믹스, 오라클 등 데이터베이스에 접근하기 위한 도구를 제공한다.
데이터 분석, 사물 인터넷	파이썬으로 만든 판다스(Pandas) 모듈을 사용하면 데이터 분석을 더 쉽고 효율적으로 할 수 있다. 필자는 아직 초보다.

할 수 있는 일이 있다면 할 수 없거나 하기 까다롭기 때문에 굳이 파이썬을 이용하지 않는 일도 있을 것이다. 대표적으로 두가지만 써보겠다.

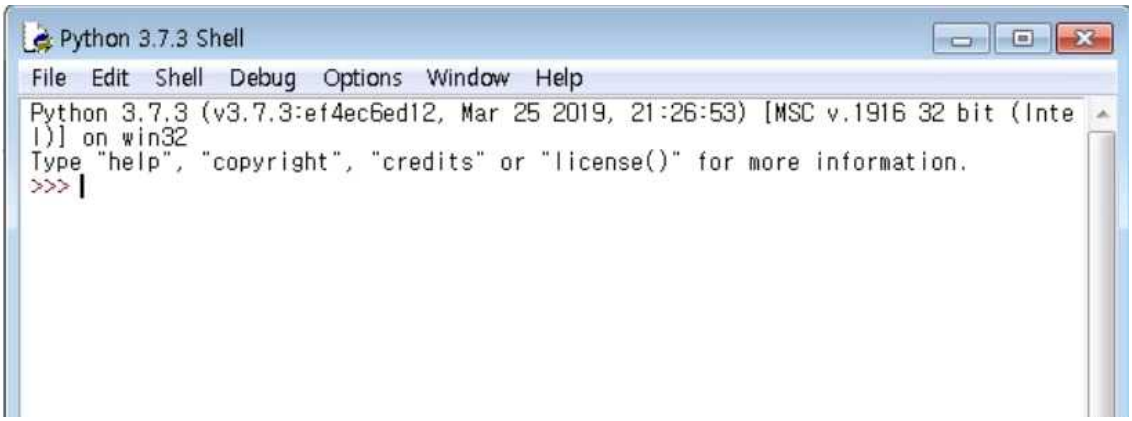
할 수 없는 일 or 까다로운 일	내용
시스템과 밀접한 프로그래밍 영역	파이썬은 대단히 빠른 속도를 요구하거나 하드웨어를 직접 건드려야 하는 프로그램에는 어울리지 않는다.
모바일 프로그래밍	아이폰 앱을 개발하는거나 안드로이드 앱을 개발하는 것은 아직 어렵다. 물론 안드로이드에서 파이썬으로 만든 프로그램이 실행 되도록 지원하기는 하지만 아직 앱을 만들기에는 역부족이다.

1-4 파이썬 설치

파이썬 설치는 배우려고 하는 사람이라면 웹을 검색해서 찾아보거나 책을 통해서 어떻게든 찾아 낼거라고 필자는 믿어 의심치 않는다. 그러므로 여기서 설치 과정은 생략하겠다. 나는 학습을 하는거지 책의 내용을 베끼는 것이나, 책을 펴내려고 하는 것이 아니기 때문이다.

1-5 파이썬과 에디터

파이썬 설치를 마쳤다면 이제 그 내용을 볼 차례다.
우선, **파이썬 IDLE**로 파이썬 프로그램을 작성해 보겠다. 작성하기 이전에 파이썬 IDLE이란 무엇일까? 정의를 보자면 '파이썬 IDLE은 파이썬 프로그램 작성을 도와주는 통합 개발 환경이다.'라고 적혀있다. 다음 화면을 보자.



파이썬 IDLE을 실행한 화면이다. 보다 시피 IDLE 셸(Shell)창이 먼저 나타난다.

IDLE은 크게 두 가지 창으로 구성된다.

1. **IDLE 셸 창** : IDLE 에디터에서 실행한 프로그램의 결과가 표시되는 창으로서 파이썬 셸과 동일한 기능을 수행한다. IDLE을 실행하면 가장 먼저 나타나는 창이다.

2. **IDLE 에디터 창** : IDLE 에디터가 실행되는 창이다.

처음 필자가 학습을 할때는 IDLE 셸 창에서 실습을 진행하였고, 점점 복잡해지고 여러 가지 코드가 늘어가면서 IDLE 에디터 창에서 실습을 진행하였다.
파이썬이 많이 부족하긴 하지만 익숙해짐도 따라서 현재는 **파이참(PyCharm)**을 이용하고 있다. 그래서 파이썬 포트폴리오에서 나는 파이참(PyCharm)을 이용할 것이다.

하지만, 그럼에도 중요한 것이 하나있다. 그것은 바로 **명령 프롬프트 창에서 프로그램을 실행**하는 것이다. 왜 명령 프롬프트 창 (command prompt) 가 중요하냐면 실제 업무에 나가면 일반적으로 명령 프롬프트 창에서 실행하기 때문이다. 이런 습관을 조금씩 들이자. 필자도 노력 중에 있다.

2장 파이썬의 기초, 자료형

참고한 책에서 이런 말이 있다. “어떤 프로그래밍 언어든 그 언어의 자료형을 알고 이해할 수 있다면 이미 그 언어의 절반을 터득한 것이나 다름없다.” 이처럼 나는 **‘2장 파이썬의 기초, 자료형’**에 최대한 많은 시간과 노력을 기울였다. 지금부터 자료형에 대하여 학습한 내용을 살펴보자.

2-1. 숫자형

숫자형이란 숫자 형태로 이루어진 자료형으로, 우리가 이미 잘 알고 있는 것이다. 컴퓨터를 조금 공부한 사람이라면 자료형 대부분에 친숙 할 거라 생각한다. 내용을 살펴보자.

항목	사용 예
정수	123, -345, 0
실수	123.45, -1234.5, 3.4e10
8진수	0o34, 0o25
16진수	0x2A, 0xFF

```
>>> a = 123
>>> a = -178
>>> a = 0
```

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a * b
12
>>> a / b
0.75
```

2-2. 문자열 자료형

문자열이란 문자, 단어 등으로 구성된 문자들의 집합을 의미한다. 문자열은 상당히 중요하다.
문자열을 표현하는 방식과 어떻게 문자열을 만드는지 내용을 살펴보자. 추가적으로 문자열에 관련된 여러 함수들의 이용도 보이겠다.

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

2-3. 리스트 자료형

지금까지 숫자와 문자열에 대해 알아보았다. 정말 기초중에 기초를 본 것이다. 이번에는 숫자의 모음이나, 집합에 관한 자료형을 살펴보자. 그 이름은 바로 '**리스트**' 자료형이다.

```
>>> a = []
>>> b = [1, 2, 3]
>>> c = ['Life', 'is', 'too', 'short']
>>> d = [1, 2, 'Life', 'is']
>>> e = [1, 2, ['Life', 'is']]
```

2-4. 튜플 자료형

튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하다 다른점을 기술하자면 다음과 같다.
(1) 리스트는 []으로 둘러싸지만 튜플은 ()으로 둘러싼다.
(2) 리스트는 그 값의 생성·삭제·수정이 가능하지만 튜플은 그 값을 바꿀 수 없다.

그렇다면 여기서 필자가 들었던 의문은 다음과 같다. “ 왜? 값을 바꿀 수 없는 것이 무엇을 의미하지? 왜? 별로 다를 것도 없는데 구분하여서 자료형을 만들었을까?” 누군가에게는 쓸데없는 질문이라고 들을수 있지만 나에게는 중요했다. 하지만 이 질문을 통해 구분을 한 가장 큰 이유에 대해 알 수 있었으므로 나에게는 좋은 질문이였다 생각한다.
그렇다면 이제 튜플을 이용하는 이유를 알아보자.
나중에 코딩을 하다보면 변경되어야 할 부분도 있고 실행을 하면서 변경이 되면 안되는 부분이 있을 것이다. 그럴 때 이용하라고 리스트와 튜플을 구분한 것이다. 답만 알았지 나도 아직 그러한 경우는 만나지 못했다.

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

2-5. 딕셔너리 자료형

딕셔너리 자료형은 대응 관계를 나타낼 수 있게 해주는 자료형이다.

딕셔너리는 Key와 Value를 한 쌍으로 같는 자료형이다. 즉, Key를 통해 Value를 얻는다. 이는 딕셔너리 자료형의 가장 큰 특징이다.

```
{Key1:Value1, Key2:Value2, Key3:Value3, ...}
```

딕셔너리 자료형의 기본 구조이다.

```
>>> a[3] = [1,2,3]
>>> a
{1: 'a', 2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

2-6. 집합 자료형

집합(Set)은 파이썬 2.3부터 지원하기 시작한 자료형이다. 왜 만들었냐하면, 집합에 관련된 것을 쉽게 처리하기 위해 만든 자료형이다.
집합 자료형에는 2가지 큰 특징이 있는데 첫 번째는 중복을 허용하지 않는것이고, 두 번째는 순서가 없다라는 것이다. 다음 자료 화면을 보자.

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

2-7. 불 자료형

불(bool) 자료형이란 참과 거짓을 나타내는 자료형이다. 설명한 것과 같이 불 자료형은 2가지 값만 가질수 있다.
(True : 참 or False : 거짓)

```
>>> a = True
>>> b = False
```

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

2-8. 변수

파이썬에서 사용하는 **변수**는 객체를 가리키는 것이라고도 말할 수 있다. 객체에 대해서는 뒤에 학습하는 부분과 겹치므로 자세히 설명이 가능한 장인 5-1에서 학습한 내용을 설명하겠다.

3장 파이썬의 구조, 제어문

3-1. if문

if문을 배우기 전에 왜? if문을 사용해야 하는지에 대해 아는 것이 중요하다.
'~고 ~하라, ~이 아니면 ~하라, ~면은 ~이다' 와 같이 이러한 내용을 프로그램에서 구현하기 위해서 if 문 즉, 제어문이 필요하다. 참이 아닐 경우에 필요한 실행할 내용을 위해 if문은 else와 같이 사용한다. 다음 그림을 보자.

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else:
    수행할 문장A
    수행할 문장B
    ...
```

조건문을 테스트하여 참이면 if문 바로 다음 문장 (if 블록)들을 수행하고, 조건문이 거짓이라면 else문 바로 다음 문장 (else 블록)들을 수행한

다. 보다시피 else문은 if문없이 단독으로 사용할수 없다.

조건문을 수행할 때 들여쓰기는 중요한 부분을 차지한다. 맨 앞장 파이썬에 관하여에서와 같이 파이썬은 들여쓰기를 소중히 해야 구조가 잘 짜여진 프로그램을 만들 수 있다.

또한, 조건문을 사용할 때 중요한 부분이 한가지 더있는데 다른 프로그래밍 언어와는 다르게 조건문 뒤에는 반드시 콜론(:)이 붙는다는 것이다. 왜 콜론이 붙는지는 귀도에게 물어봐라 필자는 그런거 모른다. 파이썬만의 독특한 방식으로 이해하고 넘어가면 좋다.
이처럼 위에 두가지 (들여쓰기, 콜론(:))를 주의하면서 조건문을 사용하도록 하자.

아래의 내용은 **if문, if~else문, if~elif~else문**을 나타낸 것이다.

1. if문

```
if 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
```

2. if~else문

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else:
    수행할 문장A
    수행할 문장B
    ...
```

3. if~elif~else 문

```
If <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
...
else:
   <수행할 문장1>
   <수행할 문장2>
   ...
```

3-2. while문

while문을 사용하는 이유는 간단하다. 반복해서 문장을 수행해야 할 경우에 사용한다. 그래서 while문을 **반복문**이라고도 부른다.

while문은 조건문이 참인 동안에 while문 아래의 문장이 반복해서 수행된다.

그렇다면 while문이 무한히 돈다면 어떻게 하겠는가? 정말 경우에따라 다르겠지만 대부분 비효율적이고 좋지 않은 코드가 생성될 것 같다. 그래서 여기 while문을 강제로 빠져나가는 방법을 소개하려 한다.

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee -1
...     print("남은 커피의 양은 %d개입니다." % coffee)
...     if coffee == 0:
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
...
```

이번에는 다른 내용을 살펴보자. 프로그램을 작성하다보면 위에경우와는 다르게 while문 내부에서 맨 처음(조건문)으로 돌아가야 하는 경우가 있을 수 있다.
이때 사용하는 것이 바로 **continue문**이다.

```
>>> a = 0
>>> while a < 10:
...     a = a + 1
...     if a % 2 == 0: continue
...     print(a)
...
1
3
5
7
9
```

앞서 while문에서 사용 할 수 있는 **break문**과 **continue문**을 살펴보았다.

이번에는 우리가 사용하는 일반 프로그램 중 많은 것들이 사용하는 무한 루프에 대해서 알아보자.
파이썬에서 무한 루프는 while문으로 구현이 가능하다. 아래의 내용은 무한 루프의 기본 형태이다. 알아두자.

3-3. for문

for문은 우선 while문과 비슷한 반복문이다. 바로 기본 구조를 살펴보자.

```
for 변수 in 리스트(또는 튜플, 문자열):
    수행할 문장1
    수행할 문장2
    ...
```

역시나 콜론(:)과 들여쓰기가 되어있다. 간단하고도 쉬운 내용이지만 항상 주의하자.

for문은 파이썬에서 range 함수와 자주 쓰는데 여기서 간단히 사용을 알고 넘어가자.

```
>>> add = 0
>>> for i in range(1, 11):
...     add = add + i
...
>>> print(add)
55
```

여기까지가 제어문의 기본적인 틀이다.
if문, while문, for문을 살펴보았고 마지막 내가 직접 코딩하는 내용들을 통해 더욱 깊이 학습할 예정이다.
여기서는 간단한 기본구조와 간단한 쓰임만 알아두면 된다.

4장 함수와 프로그램의 입력과 출력

4장에서는 지금껏 해온 것들을 바탕으로 함수, 입력과 출력, 파일 처리 방법등에 대하여 학습한 내용을 토대로 작성할 것이다.

4-1. 함수

우선 **함수**를 왜 쓰는지부터 알아야한다. 프로그래밍 언어에서 함수를 이용하는 이유는 프로그래밍을 하다보면 똑같은 내용이 반복해서 작성하고 있는 자신을 발견할때가 있다.
반복적으로 사용되는 가치있는 부분을 함수로 만들어 놓는 것이다. 또 다른 이유는 함수를 잘 사용하면 프로그램에서의 흐름을 일목요연하게 볼 수 있기 때문이다.
함수를 잘 사용하고 적절하게 만들 줄 아는 사람이 능력 있는 사람이다.

우리 함께 함수를 잘 사용하도록 노력하자.

· 함수의 구조

파이썬 함수의 구조는 아래의 내용과 같다.

```
def 함수명(매개변수):
    <수행할 문장1>
    <수행할 문장2>
    ...
```

def는 함수를 만들 때 사용하는 예약어이며, 함수 이름은 함수를 만드는 사람이 임의로 만들 수 있다.

· 매개변수와 인수

매개변수(parameter)와 **인수(arguments)**는 혼용해서 사용되는 헷갈리는 용어이므로 잘 기억해두자. 매개변수는 함수에 입력으로 전달된 값을 받는 변수를 의미하고, 인수는 함수를 호출 할 때 전달하는 입력값을 의미한다.

더욱 깊이 들여다 보자. 인수로 보내는 입력값이 몇 개가 될지 모를 때는 어떻게 해야 할까?
아래의 내용을 살펴보자.

```
def 함수이름(*매개변수):
    <수행할 문장>
    ...
```

```
>>> def add_many(*args):
...     result = 0
...     for i in args:
...         result = result + i
...     return result
...
>>>
```

매개변수 부분이 (*매개변수)로 되있는 것을 볼수 있을 것이다.
***args**처럼 매개변수 이름앞에 *을 붙이면 입력값을 전부 모아서 튜플로 만들어준다.
*앞에 이름은 본인이 원하는 이름으로 만들어 사용하면 된다.
이제 입력값이 많아서 우왕좌왕할 일이 없어서 좋다.

4-2. 사용자 입력과 출력

교수님 강의를 통해서 내가 배운 입력과 출력은 input(입력)과 print(출력)이 전부인 것 같다.
각각의 내용을 살펴보자. 우선 지금까지 설명한 내용중에 input을 사용하거나 input을 이용한 예제들이 없으므로 아래의 코드를 보자. input은 입력되는 모든 것을 문자열로 취급한다.
다음으로 print를 보겠다. 지금껏 print문이 수행해 온 일은 우리가 입력한 자료형을 출력하는 것이었다.
위에 많은 예시들에서 print 함수를 사용한 것을 알 것이다.

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3
>>> print(number)
3
>>>
```

number에는 문자열 '3'이 저장된다. 주의하자

4-3. 파일 읽고 쓰기

'**파일 읽고 쓰기**' 부분에서는 색다른 내용을 배웠다. 상당히 흥미로웠지만 어려운 부분도 많았다. 아직 많이 부족하지만 배웠던 내용을 한번 써

보겠다.
여기 부분에서는 파일을 통한 입출력 방법에 대해서 학습하였다.
학습한 내용은 파일을 새로 만든 다음 프로그램이 만든 결괏값을 새 파일에 적어봤고, 또 파일에 적은 내용을 읽고, 새로운 내용을 추가하는 방법을 알아보았다.

우선은 뭐가됐든 파일을 알아야하니깐 파일을 생성해보자.

파일 객체 = open(파일 이름, 파일 열기 모드)

파일을 생성하기 위해 파이썬 내장 함수 open을 사용했다. open 함수는 다음과 같이 '파일 이름'과 '파일 열기 모드'를 입력값으로 받고 결과값으로 파일 객체를 돌려준다.

여기에 쓴 '파일 열기 모드'에는 다음과 같은 것이 있다.

파일열기모드	설명
r	읽기모드 - 파일을 읽기만 할 때 사용
w	쓰기모드 - 파일에 내용을 쓸 때 사용
a	추가모드 - 파일의 마지막에 새로운 내용을 추가 시킬 때 사용

파일을 쓰기 모드로 열면 해당 파일이 이미 존재할 경우 원래 있던 내용이 모두 사라지고, 해당 파일이 존재하지 않으면 새로운 파일이 생성된다. open함수에 'w'가 쓰기 모드인 것을 열려준다.

그렇다면 이제 만들어진 파일에다가 내용을 써보자. 다음 화면을 살펴보자.

```
f = open("C:/doit/새파일.txt", 'w')
f.close()
```

우리가 기존에 하던 방식은 모니터 화면에 작성한 코드의 결과를 출력한 것이고, 현재 위에 있는 화면에 결과는 쓰기 모드로 열어놓은 파일에 결괏값이 적힌다는 것이다.

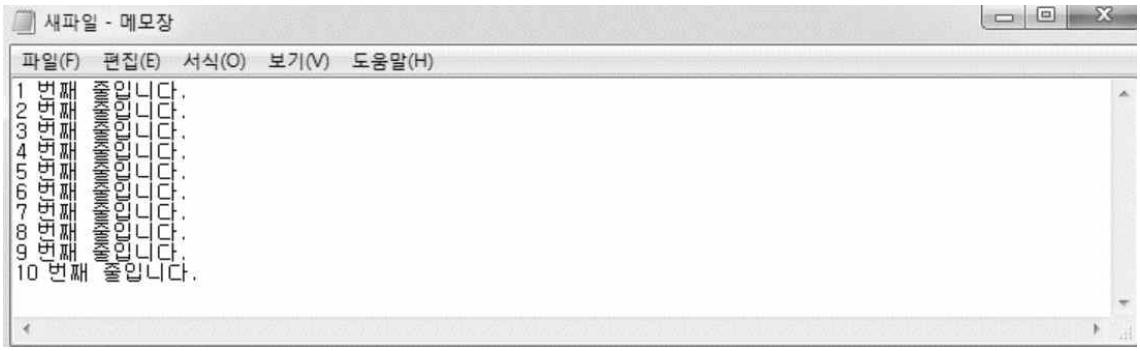
이를 위해 f.write (data)라 써진 부분이 적힌 것이다. 매우 흥미롭지 않은가?

여태껏 다른 프로그래밍 언어를 배우면서 알지 못했는데, 파이썬에서 파일에 관한 내용을 알게되어 무척 기쁘다.

이제는 앞장에서 언급했던 **명령 프롬프트 창**에서 예제를 실행시켜 보자.

```
C:\Users> cd C:\doit
C:\doit>python writedata.py
C:\doit>
```

마지막으로 생성된 파일에 담겨져 있는 내용이 어떤지 살펴보자.



지금까지 **'파일 열기 모드'**에서 **'쓰기 모드 - w'**를 알아봤다 '파일 열기 모드'의 **'읽기 모드 - r'**, ' **추가 모드 - a'**에 대해서도 알아보자.
이 부분에서는 프로그램의 외부에 저장된 파일을 읽는 여러 가지 방법에 대해 학습한 내용을 기술할 것이다.

그리고 기존에 '쓰기 모드'로 파일을 열 때 이미 존재하는 파일의 내용이 모두 사라지는 점을 고려해서 '추가 모드'를 이용해 기존에 있던 값은 유지하면서 새로운 값을 추가하는 경우에 대해서도 보일 것이다.

· **읽기 모드 ('r')** - readline 함수와 readlines 함수, read 함수 사용

1. readline

```
# readline_test.py
f = open("C:/doit/새파일.txt", 'r')
line = f.readline()
print(line)
f.close()
```

2. readlines

```
f = open("C:/doit/새파일.txt", 'r')
lines = f.readlines()
for line in lines:
    print(line)
f.close()
```

3. read

```
f = open("C:/doit/새파일.txt", 'r')
data = f.read()
print(data)
f.close()
```

· **쓰기 모드 ('a')** - 파일에 새로운 내용 추가하기

```
# adddata.py
f = open("C:/doit/새파일.txt",'a')
for i in range(11, 20):
    data = "%d번째 줄입니다.\n" % i
    f.write(data)
f.close()
```

결과는 다음과 같다.

5장 파이썬 심화 과정

1장에서 4장까지 오면서 파이썬에서 사용하는 기본적인 것들은 거의 알아봤다 이번 장에서는 심화 과정 클래스, 모듈, 패키지, 내장 함수에 대하여 이야기 해보겠다.

5-1. 클래스

항상 시작을 **왜? 필요한지**에 대해 밝히고 시작하겠다. 파이썬으로 프로그램을 작성할 때 클래스를 이용하지 않고 만든 프로그램은 많다. 나는 클래스가 필요한 이유에 대해서 대답하자면 클래스로 만든 객체는 객체마다 고유한 성격을 가진다는 것으로 답할 것이다. 서로에게 영향을 주지 않는다는 뜻이다.
클래스가 있으면 따라오는 것이 있다. 그것은 바로 메서드 이다. 클래스 안에서 구현된 함수를 다른 말로 **메서드(Method)**라고 부른다. 그렇다면 간단한 **클래스와 메서드의 구조**를 살펴보자.

```
>>> class FourCal:
...     def setdata(self, first, second):
...         self.first = first
...         self.second = second
...
>>>
```

내용을 보자니 이상한 것이 보인다. 필자도 처음에는 “이게 뭐지?” 라 생각했지만 책의 훌륭한 설명 덕분에 이해하고 넘어갔다. 함수를 구현한 부분에 매개변수로 ‘self’라는 값이 있다.
결론을 말하자면 파이썬 메서드의 관례적으로 구현하는 독특한 파이썬만의 특징이다. 함수를 호출한 객체 자신을 나타내는 말이라 이해하면 좋다.
객체 이야기가 나왔으니 객체변수에 대한 이야기도 해보겠다. 객체변수란 객체에 생성되는 객체만의 변수를 말한다. 더욱 자세히 알아보자.

객체변수는 그 객체의 고유 값을 저장할 수 있는 공간이다. 객체변수는 다른 객체들 영향을 받지 않고 독립적으로 그 값을 유지한다는 점을 꼭꼭 기억해야한다. 아래의 간단한 내용을 보고 이해해보면 될 거 같다.

```
def setdata(self, first, second):   # ① 메서드의 매개변수
    self.first = first              # ② 메서드의 수행문
    self.second = second            # ③ 메서드의 수행문
```

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(4, 2)
>>> b.setdata(3, 7)
>>> id(a.first)   # a의 first 주소값을 확인
1839194944
>>> id(b.first)   # b의 first 주소값을 확인
1839194928
```

객체가 무엇이고 객체 변수가 무엇인지 알아보았다. 위의 방법은 객체를 생성한 후에 매개변수에 맞게 값을 전달해서 객체변수에 값을 저장하는 방식으로 초기값을 설정한 것이다.

그렇다면 이 부분에서는 **'생성자'**라는 것을 이용해서 객체를 생성함과 동시에 초깃값을 바로 설정하는 법에대해 배운 내용을 적어보겠다.
이는 객체에 초깃값을 설정해야할 필요가 있을 때 사용하면 된다. 먼저 생성자를 클래스 내의 함수 이름을 **__init__** 써서 구현해 준 후 매개변수 값은 본인이 구현하고자 하는 내용으로 만들어주면 된다.
이렇게 함수 이름에 __init__을 붙이면 생성자로 인식되어 객체가 생성되는 시점에 자동으로 호출되는 차이가 생긴다.

```
def __init__(self, first, second):
    self.first = first
    self.second = second
```

마지막으로, 클래스의 상속과 메서드 오버라이딩에 대하여 학습한 내용을 정리해보자.
클래스 상속은 기존 클래스가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이라면 상속을 사용해야 한다. 그렇다면 상속이라는 것이 무엇일까? **상속(Inheritance)**이란 어떤 클래스를 만들 때 다른 클래스의 기능을 물려받을 수 있게 만드는 것이다.

class 클래스 이름 (상속할 클래스 이름)

상속은 위와 같은 형태로 사용하면 된다.

메서드 오버라이딩은 부모 클래스에 있는 메서드를 동일한 이름으로 다시 만드는 것이다.

5-2. 모듈

'모듈'이란 함수나 변수 또는 클래스를 모아 놓은 파일이다. 다른 파이썬

프로그램을 불러와 사용할 수 있게끔 만든 파이썬 파일이라고 할 수 있다.
그렇다면 모듈을 어떻게 파이썬에서 불러와 사용할 수 있을까? 이는 import를 이용하면 된다. **import**는 이미 만들어 놓은 파이썬 모듈을 사용할 수 있게 해주는 명령어이다.
import를 사용할때는 다음 사항을 알아둬야한다. 현재 디렉터리에 있는 파일이나 파이썬 라이브러리가 저장된 디렉터리에 있는 모듈만 불러올 수 있다는 것을 알아둬야한다.

5-3. 내장 함수

		내장 함수		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

(출처 : https://docs.python.org/ko/3/library/functions.html#built-in-functions)

파이썬 내장함수는 이렇게 무수히 많다. 이 내용을 전부다 알아볼수는 없으니 가벼운 마음으로 자주 사용되는 함수 위주로 살펴보려 한다. abc

순서대로 학습한 내용을 기술해 보겠다.

(출처 : https://wikidocs.net/book/1) (점프 투 파이썬 - 박응용)

1. **abs(x)**는 어떤 숫자를 입력받았을 때, 그 숫자의 절댓값을 돌려주는 함수이다.

```
>>> abs(3)
3
>>> abs(-3)
3
>>> abs(-1.2)
1.2
```

2. **any(x)**는 x 중 하나라도 참이 있으면 True를 돌려주고, x가 모두 거짓일 때에만 False를 돌려준다.

```
>>> any([1, 2, 3, 0])
True
```

```
>>> any([0, ""])
False
```

3. **chr(i)**는 아스키(ASCII) 코드 값을 입력받아 그 코드에 해당하는 문자를 출력하는 함수이다.

```
>>> chr(97)
'a'
>>> chr(48)
'0'
```

4. **dir**은 객체가 자체적으로 가지고 있는 변수나 함수를 보여 준다. 다음 예는 리스트와 딕셔너리 객체 관련 함수(메서드)를 보여 주는 예이다.

```
>>> dir([1, 2, 3])
['append', 'count', 'extend', 'index', 'insert', 'pop',...]
>>> dir({'1':'a'})
['clear', 'copy', 'get', 'has_key', 'items', 'keys',...]
```

5. **divmod(a, b)**는 2개의 숫자를 입력으로 받는다. 그리고 a를 b로 나눈 몫과 나머지를 튜플 형태로 돌려주는 함수이다.

```
>>> divmod(7, 3)
(2, 1)
```

내용을 보면은 7을 3으로 나눈 몫인 2와 나머지 1이 튜플형태로 돌려준 것을 볼 수있다.

```
>>> 7 // 3
2
>>> 7 % 3
1
```

직접 '//'와 '%'을 이용해서 비교해보자.

6. **eval(expression)**은 실행 가능한 문자열(1+2, 'hi' + 'a' 같은 것)을 입력으로 받아 문자열을 실행한 결괏값을 돌려주는 함수이다.

```
>>> eval('1+2')
3
>>> eval("'hi' + 'a'")
'hia'
>>> eval('divmod(4, 3)')
(1, 1)
```

7. **hex(x)**는 정수 값을 입력받아 16진수(hexadecimal)로 변환하여 돌려주는 함수이다.

```
>>> hex(234)
'0xea'
>>> hex(3)
'0x3'
```

8. **id(object)**는 객체를 입력받아 객체의 고유 주소 값(레퍼런스)을 돌려주는 함수이다.

```
>>> a = 3
>>> id(3)
135072304
>>> id(a)
135072304
>>> b = a
>>> id(b)
135072304
```

'3'과 ,a, b 모두 주소 값이 같은 것을 직접 확인 가능하다.

9. **int(x)**는 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 돌려주는 함수로, 정수를 입력으로 받으면 그대로 돌려준다.

```
>>> int('3')
3
>>> int(3.4)
3
```

10. **isinstance(object, class)**는 첫 번째 인수로 인스턴스, 두 번째 인수로 클래스 이름을 받는다. 입력으로 받은 인스턴스가 그 클래스의 인스턴스인지를 판단하여 참이면 True, 거짓이면 False를 돌려준다.

```
>>> class Person: pass
...
>>> a = Person()
>>> isinstance(a, Person)
True
```

'a'인스턴트가 Person클래스의 인스턴트인지 확인이 완료돼 true값을 반환해줬다.

```
>>> b = 3
>>> isinstance(b, Person)
False
```

'b'인스턴트는 Person클래스의 인스터트가 아니여서 False값이 반환되었다.

11. **len(s)**은 입력값 s의 길이(요소의 전체 개수)를 돌려주는 함수이다.

```
>>> len("python")
6
>>> len([1,2,3])
3
>>> len((1, 'a'))
2
```

12. **list(s)**는 반복 가능한 자료형 s를 입력받아 리스트로 만들어 돌려주는 함수이다.

```
>>> list("python")
['p', 'y', 't', 'h', 'o', 'n']
>>> list((1,2,3))
[1, 2, 3]
```

```
>>> a = [1, 2, 3]
>>> b = list(a)
>>> b
[1, 2, 3]
```

'b'에 다가 리스트 a를 복사 하기도 가능하다.

13. **map(f, iterable)**은 함수(f)와 반복 가능한(iterable) 자료형을 입력으로 받는다. map은 입력받은 자료형의 각 요소를 함수 f가 수행한 결과를 묶어서 돌려주는 함수이다.

```
>>> def two_times(x):
...     return x*2
...
>>> list(map(two_times, [1, 2, 3, 4]))
[2, 4, 6, 8]
```

14. **max(iterable)**는 인수로 반복 가능한 자료형을 입력받아 그 최댓값을 돌려주는 함수이다.

```
>>> max([1, 2, 3])
3
>>> max("python")
'y'
```

15. **range([start,] stop [,step])**는 for문과 함께 자주 사용하는 함수이다. 이 함수는 입력받은 숫자에 해당하는 범위 값을 반복 가능한 객체로 만들어 돌려준다.

```
>>> list(range(5))
[0, 1, 2, 3, 4]
```

이처럼 하나의 숫자만 입력하면 0부터 시작해 입력받은 숫자 -1 까지 범위가 생긴다.

```
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> list(range(0, -10, -1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
```

range의 맨 마지막 매개변수는 간격을 의미하는데 얼마의 간격을 설정해서 값의 범위를 정하는지 선택할 수 있다. 1번 예제는 양의 방향으로 '2'만큼, 2번 예제는 음의 방향 '-1'만큼 범위를 정해 리스트 함수를 이용해 값을 저장한 것을 보여준다.

16. **sorted(iterable)** 함수는 입력값을 정렬한 후 그 결과를 리스트로 돌려주는 함수이다.

```
>>> sorted([3, 1, 2])
[1, 2, 3]
>>> sorted(['a', 'c', 'b'])
['a', 'b', 'c']
>>> sorted("zero")
['e', 'o', 'r', 'z']
>>> sorted((3, 2, 1))
[1, 2, 3]
```

리스트 자료형에도 sort 함수가 있다. 하지만 리스트 자료형의 sort 함수는 리스트 객체 그 자체를 정렬만 할 뿐 정렬된 결과를 돌려주지는 않는다.

17. **tuple(iterable)**은 반복 가능한 자료형을 입력받아 튜플 형태로 바꾸어 돌려주는 함수이다. 만약 튜플이 입력으로 들어오면 그대로 돌려준다.

```
>>> tuple("abc")
('a', 'b', 'c')
>>> tuple([1, 2, 3])
(1, 2, 3)
>>> tuple((1, 2, 3))
(1, 2, 3)
```

18. **type(object)**은 입력값의 자료형이 무엇인지 알려 주는 함수이다. 중요한 함수이니 기억하자!

```
>>> type("abc")
<class 'str'>
>>> type([ ])
<class 'list'>
>>> type(open("test", 'w'))
<class '_io.TextIOWrapper'>
```

6장 연습 문제

파이썬 1학기가 끝나기 전에 내가 배운 내용은 거의 다 적은 것 같다.

물론 부분마다 생략된 부분과 내가 미처 생각지 못한 부분은 있겠지만, 여러번 검토를 하고 계속 책과 인터넷을 통해 학습을 하면서 수정하고 삭제하고 했으니 꽤 괜찮은 파이썬 포트폴리오가 만들어진 것 같다.

하지만, 아직 완성이 덜 되었다. 위에 내용들을 하나하나 살펴본다면 기본적인 내용만 있고 해당 부분에 더욱 깊게 들어가는 코드나 예제들이 부족하다는 것을 느낄 수도 있다.

그래서 이번 5-4장에서는 연습 문제를 통해서 내가 위에서 학습한 내용들을 총 정리하면서 직접 코딩하고 실행하고, 실패하고, 성공하는 문제를 고스란히 올리려고 한다. 물론 여기는 성공한 것만 올라올 것이다.

1. split함수와 join함수를 이용해 ':'을 '#'으로 대체

```
#01 번 문제
a="a:b:c:d"
b= a.split(":")
print(b)
c = '#'.join(b)
print(c)
```

```
['a', 'b', 'c', 'd']
a#b#c#d
```

2. 딕셔너리 안에 없는 값이 있을 경우 get함수를 이용해 디폴트 값을 주는 예제

```
>>> a = {'A':90, 'B':80}
>>> a.get('c', 70)
70
>>>
```

3. a,b의 리스트를 생성하고 기존 리스트에 일반적인 값을 넣었을 경우

와 extend함수를 사용했을 때 주소의 값이 어떻게 달라지는지 확인

```
>>> a = [1,2,3]
>>> b=[1,2,3]
>>> id(a)
2881546048008
>>> id(b)
2881546048520
>>> a = a+[4,5]
>>> b.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4, 5]
>>> id(a)
2881547541640
>>> id(b)
2881546048520
>>>
```

4. 하나의 리스트를 생성하여 50넘는 값들을 if문으로 걸러낸 후 더하기

```
#04번 문제
A = [20,22,25,60,40,70,80,99]

result = 0
while A:
    mark = A.pop()
    if mark >=50:
        result += mark

print(result)
```

```
309
```

5. input을 이용해 숫자를 입력받고 입력받은 문자열을 int 자료형으로 변환하여 입력받은 숫자들로 합을 나타냄.

```python
a = input("숫자를 입력하세요: ")
numbers = a.split(",")
total = 0
for n in numbers:
    total += int(n)    # 입력은 문자열이므로 숫자로 변환해야 한다.
print(total)
```

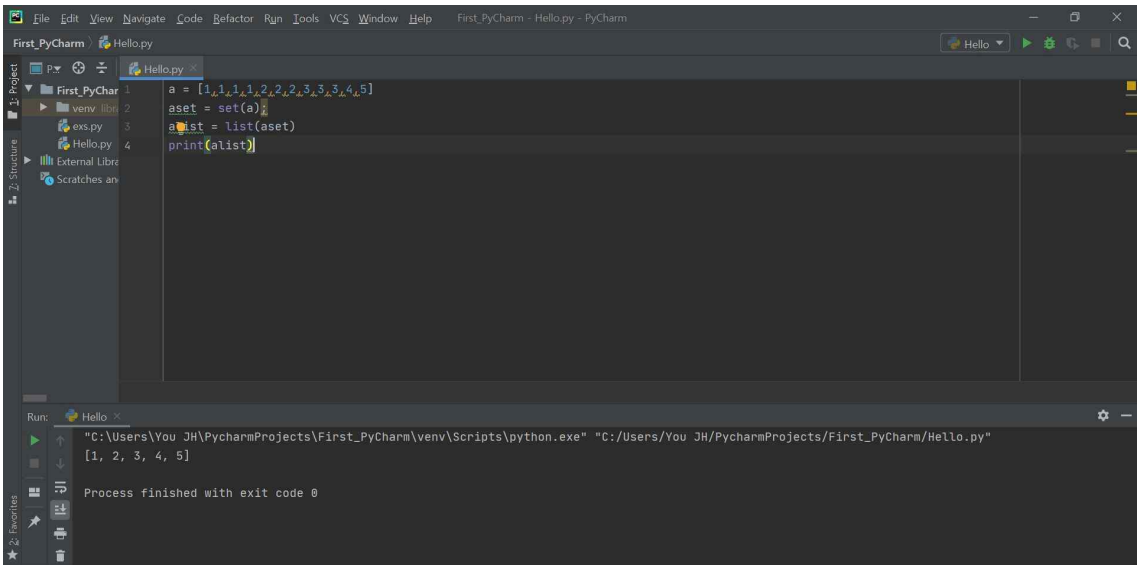
```
숫자를 입력하세요: 1,2,3,4,5,10
25
```

6. 기본적인 for문을 이용해 간단한 구구단을 만든 모습

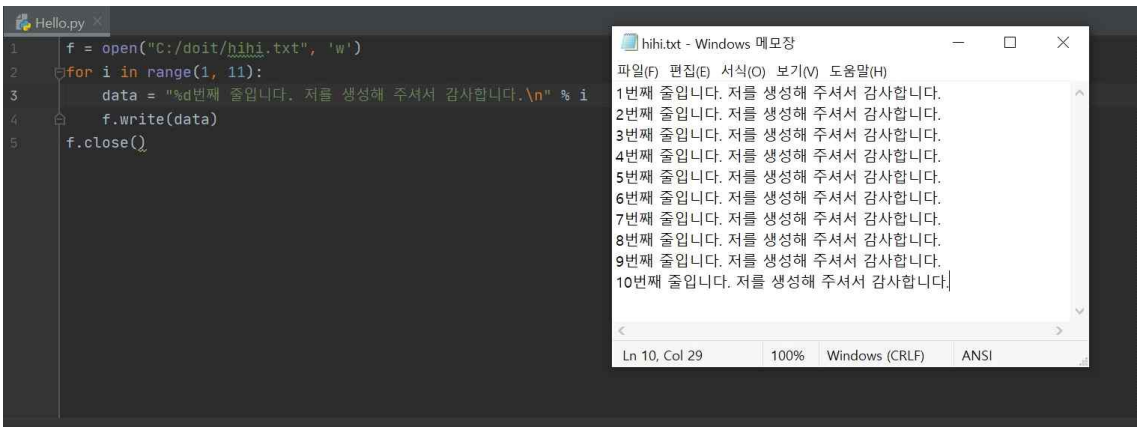
```python
#구구단 예제
for i in range(2,10):
    for j in range(1,10):
        print(i*j, end=" ")
```

```
"C:\Users\You JH\PycharmProjects\First_PyCharm\venv\Scripts\python.exe" "C:/Users/You JH/PycharmProjects/First_Py
2 4 6 8 10 12 14 16 18 3 6 9 12 15 18 21 24 27 4 8 12 16 20 24 28 32 36 5 10 15 20 25 30 35 40 45 6 12 18 24 30 3
Process finished with exit code 0
```

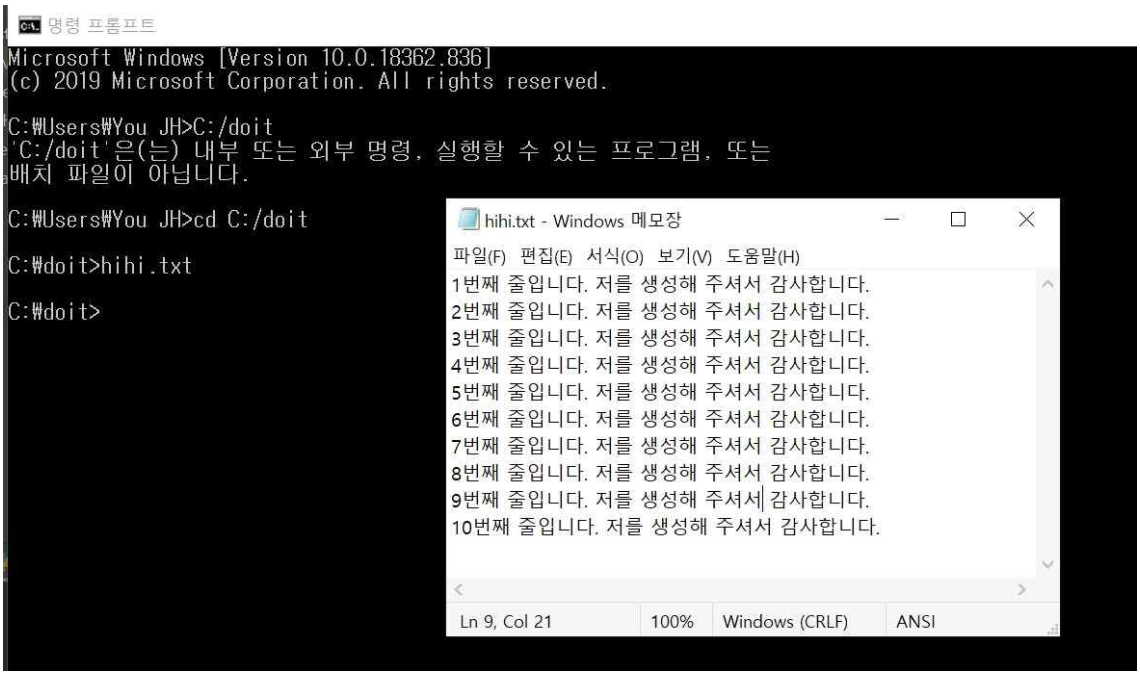
7. 집합 자료형과 리스트 자료형을 이용해 리스트안에 중복되는 값들을 정리하는 방법.



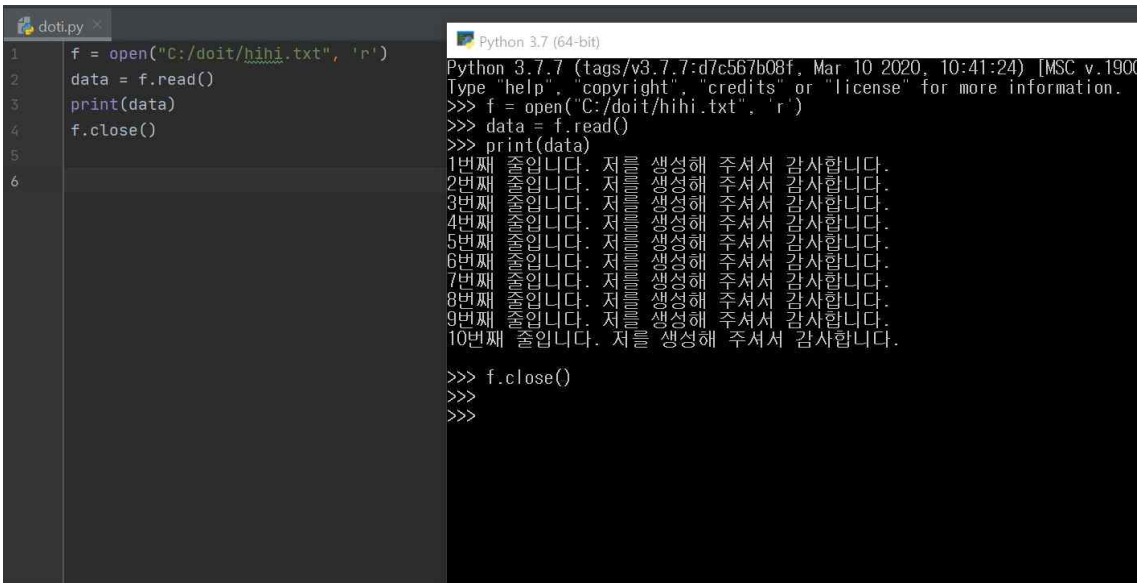
8. 파이참에서 텍스트 문서에 open함수를 이용해 값 입력하기.



9. 명령 프롬프트에서 저장된 텍스트문서 확인.



10. IDLE 창에서 read함수를 이용해 저장된 텍스트 값 읽기.



11. 마지막으로 사칙연산 클래스를 구연하고 간단한 계산을 할 수 있도록 만들어 보겠다. 하지만, 내가 작성한 코드를 보면 간단하게 못한걸 볼 수 있고, 상당히 인간적이다.

```
>>> class FourCal:
        def setdata(self, first, second):
            self.first = first
            self.second = second
        def add(self):
            result = self.first+self.second
            return result
        def sub(self):
            result = self.first-self.second
            return result
        def mul(self):
            result = self.first*self.second
            return result
        def div(self):
            result = self.first/self.second
            return result
```

인터프리터를 이용해 클래스를 구현해봤다. 클래스 내의 메소드는 데이터를 지정해주는 setdata와 나머지는 차례대로 더하기, 빼기, 곱하기, 나누기이다.

그리고 상속을 받아 새로운 메소드를 추가 해봤다. 그리고 다음 내용은 내가 틀리고 맞게 한 내용들이다. 이 위에 코드들도 마찬가지로 이와 동일한 과정을 거쳐서 현재 포트폴리오를 완성했다.

```
>>> a = FourCal()
>>> b = FourCal()
>>> a.setdata(10,20)
>>> b.setdata(30,40)
>>> print("%d %d" % a,b)
Traceback (most recent call last):
  File "<pyshell#27>", line 1, in <module>
    print("%d %d" % a,b)
TypeError: %d format: a number is required, not FourCal
>>> print(a);
<__main__.FourCal object at 0x000002864B3C5C08>
>>> a.first
10
>>> a.second
20
>>> add(a,b)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    add(a,b)
NameError: name 'add' is not defined
>>> add(a.first, b.second)
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    add(a.first, b.second)
NameError: name 'add' is not defined
>>> a.add()
30
>>> b.sub
<bound method FourCal.sub of <__main__.FourCal object at 0x000002864B23C9C8>>
>>> b.sub()
-10
>>> class PowFourCal(FourCal):
        def pow(self):
                result = self.first**self.second
                return result


>>> a = PowFourCal()
>>> a.setdata(30,50)
>>> a.pow()
717897987691852588770249000000000000000000000000000000000000000000000000000000000000
>>> a.setdata(3,5)
>>> a.pow()
243
>>>
```

이상으로 파이썬 포트폴리오를 마친다.

파이썬 포트폴리오 작성 후기 & 느낌

후기	파이썬 포트폴리오를 드디어 끝냈다. 교수님께서 정말 오래전에 내주신 것 같은데 말씀하신데로 상당히 오랜 시간과 노력이 필요했다. 학습을 하면서 막히는 부분도 있었고 잘되는 부분도 있었지만, 교수님의 강의와 책, 그리고 다른 교재와 인터넷을 통해서 부족한 부분을 채워나가면서 하루하루 최소한의 약진을 한 듯 하다. 어찌어찌 40 쪽을 맞췄는데 처음에 계획했던 페이지 수는 30페이지였다. 지금 후기를 쓰는것도 “아, 드디어 지긋지긋한 파이썬 포트폴리오가 끝났네.”가 아니라 뭐가 만들어진 것 같고 뿌듯하기도 하고 약간의 성취감이 들기도 하다. 처음에는 막막했지만 완성해놓고 보니 “아, 내가 그래도 이걸 완성을 하고 후기를 작성하고 있구나. 좋은 경험이야.”라고 말을 할 수 있다.
느낌	이렇게 학습한 것을 직접 작성하고 생각하고 구성을 짜고, 한 것은 초,중,고,대학을 통들어 처음이였다. 많이 힘들고 좌절도 했지만 ‘나는 안되면 더해’ 라는 마음가짐을 가지고 끝까지 완성했다. 물론 부족한 부분도 있겠지만, 하루하루 발전하는 모습을 보이고싶다. 재미있었고 힘든 경험이였다. 다음에 한번더 한다면 ... 글쎄라는 생각이 들거같긴 하지만, 확실히 신선한 경험이였던 것은 분명하다.
노력할 점	학습을 하면서 예제를 많이 작성하지 못했다. 조금더 예제나 내가 생각한 부족한 코딩 부분에 많은 집중을 해야겠다.