나는 초보 개발자이고, 평소에 좋은 코드를 작성하고 있다고 생각하지는 않는다.
다만 나보다 프로그래밍에 익숙지 않은 사람의 시선에 맞춰 좋은 코드 작성법을 가르쳐줄 정도의 깜냥은 있다고 생각한다.
그래서 이 글을 쓴다.

평소에 작성하던 파이썬 코드를 좀 더 좋은 코드로 진화시켜 보자.
목진화 아님.

사진 출처

내가 프로그래밍을 처음 배웠을 때는, 코드의 가독성이나 규칙성 등등 다 제쳐 놓고 일단 구현하는 데 집중했다.
그래야 비슷하게나마 구현할 수 있었으니까.

내 프로그래밍 실력은 이제 막 초보 딱지를 뗀 정도라고 생각한다.
그럼에도 불구하고, 슬슬 코드를 보면 아쉬운 점들이 보인다.
불필요한 반복이 보이고, 일관되지 않은 형식이 보인다.
그 시선을 이 글에 녹여내보고자 한다.

본격적인 시작에 앞서 질문 하나를 던져본다.
좋은 코드란 무엇일까?
사람마다 생각이 다르겠지만, 내 생각에는 이렇다.

우선 읽기 쉬워야 한다.
그래서 이해하기 쉬워야 한다.
역할별로 분리가 명확하게 되어 있어 변경이나 재사용이 쉬워야 한다.
불필요한 반복 없이 효율적이어야 한다.

말로 늘어놓으니 복잡한데, 좋은 코드는 깔끔한 집과 유사하다.
신발은 신발장에, 음식은 냉장고에 넣듯이 코드도 용도와 쓰임에 맞게 일정한 규칙성을 가지고 분리한다.
어느 위치에 어떤 물건이 있는지 알면 나중에 찾거나 새 물건을 가져다 놓기 쉽다.
나만의 집을 꾸미듯 좋은 코드를 작성해 보자.

지식인에서 막 답변하고 온 아래 문제를 예로 들어보자.

리스트 과제

1. 메뉴를 이용하여 리스트에 친구 이름관리
   - mylist : 친구 이름을 저장할 리스트
   - 메뉴

```
**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 1
```

: 잘못된 번호 입력시 오류메시지 출력 후 다시 실행

   - 이름추가
     - 이름 입력 받고 새 이름이면 리스트에 추가하고 리스트 출력 후 메뉴
     - 있는 이름이면 "이미 있는 이름" 을 출력하고 다시 메뉴로 이동
   - 이름 삭제
     - 이름 입력 받고 있는 이름이면 리스트에서 삭제 후 리스트 출력 후 메뉴
     - 없는 이름이면 "해당 이름 없음 " 출력 후 다시 메뉴로 이동
   - 이름 수정
     - 이름 입력 받고 있는 이름이면 새로운 이름을 입력 받아 리스트에서 이름을 변경하고 리스트 출력 (같은 위치에 저장 되어야함) 후 메뉴
     - 없는 이름이면 "해당 이름 없음 " 출력 후 다시 메뉴로 이동

1

```
**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 1
이름 : 김철수
['김철수']

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 1
이름 : 김철수
이미 있는 이름

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 1
이름 : 홍길동
['김철수', '홍길동']

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 1
이름 : 이영희
['김철수', '홍길동', '이영희']
```

```
**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 2
이름 : 오현수
해당 이름은 없음

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 2
이름 : 홍길동
['김철수', '이영희']

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 3
이름 : 홍길동
해당 이름은 없음
```

```
**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 3
이름 : 이영희
새이름 : 이정희
['김철수', '이정희']

**********
1. 이름 추가
2. 이름 삭제
3. 이름 수정
4. 종료
메뉴 선택 : 4
>>>
```

2

좋은 코드고 뭐고 일단 작동하는 것이 먼저다.

조건에 맞게 구현부터 해보았다.

코드가 좀 길다.

참고로 코드 테스트는 통합 개발 환경 repilt.com의 Python 3.8.12 버전으로 진행하였다.

```
mylist = []

def menu():
    print()
    print('*' * 19)
    print('1. 이름 추가')
    print('2. 이름 삭제')
    print('3. 이름 수정')
    print('4. 종료')
    n = int(input('메뉴 선택 : '))
    if n == 1:
        name = input('이름 : ')
        if name not in mylist:
            mylist.append(name)
            print(mylist)
        else:
            print('이미 있는 이름')
        menu()
    elif n == 2:
        name = input('이름 : ')
        if name not in mylist:
            print('해당 이름은 없음')
        else:
            mylist.remove(name)
            print(mylist)
        menu()
    elif n == 3:
        name = input('이름 : ')
        if name not in mylist:
            print('해당 이름은 없음')
        else:
            new_name = input('새 이름 : ')
            i = mylist.index(name)
            mylist[i] = new_name
            print(mylist)
        menu()
    elif n == 4:
        return
    else:
        print('잘못된 번호 입력')
        menu()

menu()
```

아마 이렇게 제출하더라도 과제는 통과할 수 있을 것이다.
그래도 좀 더 욕심을 내서, 여기서 어떤 점을 고치면 좋을지 생각해 보자.
떠오르는 것 중 우선순위가 높다고 생각하는 것부터 순서대로 적어봤다.

1. 에러로 인한 프로그램 강제 종료 방지하기
2. 함수 호출 횟수 줄이기
3. 유사하게 반복되는 부분 합치기
4. 상수 분리하기

하나씩 순서대로 고쳐보자.

▶ 에러로 인한 프로그램 강제 종료 방지하기

프로그램의 예외 처리는 중요하다.
당장 100줄 안 되는 소스 코드 실행하다 콘솔 창에 에러 메시지가 나와도 검색 엔진의 힘을 빌리는데, 1000줄이나 10000줄이라면?
무엇 때문에 에러가 발생했는지 찾는 것부터 고역이다.
빨리 에러를 해결하기 위해서는, 어디서 에러가 발생했는지 찾기 쉽게 처리해 주는 것이 중요하다.

무엇보다도 예외 처리가 제대로 되지 않은 프로그램은 강제 종료될 수 있다.
게임에 몰입하던 도중 게임 프로그램이 종료된 경험이 있다면, 실행 중인 프로그램이 강제 종료되는 것이 얼마나 무시무시한 일인지 알 것이다.
물론 프로그램마다 조금씩 다르겠지만 일반적으로 프로그램은 에러가 발생하더라도 강제로 종료되지 않고, 에러 코드 등 특정 신호를 보내 에러의 유형과 발생 원인을 파악할 수 있게끔 만들어지는 것이 좋다.

설명이 길었는데, 그렇다면 위 코드에서 강제 종료가 발생할 만한 부분은 어디일까?
바로 입력 부분이다.

```
n = int(input('메뉴 선택 : '))
```

여기서 만약 입력으로 정수 변환이 불가능한 실수, 알파벳, 한글 등이 들어온다면?
프로그램은 ValueError가 발생하여 강제 종료된다.

나는 이를 막기 위해 입력을 받은 값이 숫자 문자열일 때만 정수형으로 변환하도록 수정했다.

```
n = input('메뉴 선택 : ')
if not n.isdigit():
    print('잘못된 입력')
    menu()
else:
    n = int(n)
    # ...
```

▶ 함수 호출 횟수 줄이기

함수를 많이 호출할수록 프로그램의 처리 속도는 느려진다.
그래서 가능하다면, 함수는 가급적 적게 호출하는 것이 좋다.
menu()의 초반부 코드를 살펴보자.

```
print()
print('*' * 19)
print('1. 이름 추가')
print('2. 이름 삭제')
print('3. 이름 수정')
print('4. 종료')
```

print()를 6번이나 호출하고 있다.
사실 이 경우 1번 호출이나 6번 호출이나 크게 속도 차이가 나지 않을 것 같긴 하지만, 굳이 1번이면 될 것을 6번 호출할 이유도 없다.
출력하는 문자열을 합쳐서, 출력 결과는 유지하되 print() 호출 한 번으로 해결할 수 있게끔 수정해 보자.

```
print('\n' + '*' * 19 + '\n1. 이름 추가' + '\n2. 이름 삭제' + '\n3. 이름 수정' + '\n4. 종료
```

'₩n'은 개행문자라고 하는데, 쉽게 말해 엔터 한 번과 동일한 역할을 한다.
처음보다 가독성이 떨어지긴 해도, 함수 호출 횟수를 5번이나 줄였다.
만약 가독성을 지키기 위해 줄 수를 유지하고 싶다면 아래와 같은 형태로 수정하는 것도 가능하다.

```
print('\n'
      + '*' * 19
      + '\n1. 이름 추가'
      + '\n2. 이름 삭제'
      + '\n3. 이름 수정'
      + '\n4. 종료')
```

▶ 유사하게 반복되는 부분 합치기

아래 조건문을 살펴보자.

```
if n == 1:
    name = input('이름 : ')
    if name not in mylist:
        mylist.append(name)
        print(mylist)
    else:
        print('이미 있는 이름')
    menu()
elif n == 2:
    name = input('이름 : ')
    if name not in mylist:
        print('해당 이름은 없음')
    else:
        mylist.remove(name)
        print(mylist)
    menu()
elif n == 3:
    name = input('이름 : ')
    if name not in mylist:
        print('해당 이름은 없음')
    else:
        new_name = input('새 이름 : ')
        i = mylist.index(name)
        mylist[i] = new_name
        print(mylist)
    menu()
elif n == 4:
    return
else:
    print('잘못된 번호 입력')
    menu()
```

어째 전반적으로 형식이 좀 비슷하다.
n이 2, 3인 경우에는 이름을 입력받고, 이름이 mylist에 있느냐 없느냐에 따라 특정 동작을 수행한다.
n이 4인 경우를 제외하면 모두 마지막에 menu()를 호출한다.
반복되는 부분을 합쳐보자.

```
if n == 1:
    name = input('이름 : ')
    if name not in mylist:
        mylist.append(name)
        print(mylist)
    else:
        print('이미 있는 이름')
elif 1 < n < 4:
    name = input('이름 : ')
    if name not in mylist:
        print('해당 이름은 없음')
    else:
        if n == 2:
            mylist.remove(name)
        else:
            new_name = input('새 이름 : ')
            i = mylist.index(name)
            mylist[i] = new_name
        print(mylist)
elif n == 4:
    return
else:
    print('잘못된 번호 입력')
menu()
```

코드가 좀 더 짧아졌다.
다만 여기서 주의해야 할 점이 있다.
코드를 줄이다가 실수로 이전과 다른 결과가 나오는 프로그램이 되지 않도록 신경 써야 한다.
보기 좋은 떡, 먹기 좋은 맛있는 떡이 되기 전에 일단 먹을 수 있는 떡이 되는 게 먼저다.
또한 코드를 줄이는 것이 가독성이나 확장성을 과하게 줄이지는 않는지도 고민해야 한다.

▶ 상수 구분하기

일단 프로그램이 실행되면 값이 변하지 않는 변수를 상수라고 하는데, 이러한 조건을 갖춘 변수는 상수처럼 구분해 주는 것이 좋다.
파이썬 프로그래밍을 하다 보면 url, api key와 같은 특정 변수의 값만 바꿔서 여러 번 실행하는 경우가 생긴다.
그때마다 일일이 찾아서 수정하는 일이 없도록, 상수는 모아서 코드 상단에 두거나 별도의 파일로 분리해두면 좋다.
다만 파이썬에는 상수 키워드가 없기 때문에, 대문자와 밑줄만 이용해 변수 이름을 지어서 상수처럼 보이게 한다.

```
""" CONST """
NEW_LINE_MSG = '\n'
SEP_STAR_MSG = '*'
OPTION_1_MSG = '1. 이름 추가'
OPTION_2_MSG = '2. 이름 삭제'
OPTION_3_MSG = '3. 이름 수정'
OPTION_4_MSG = '4. 종료'
SELECT_O_MSG = '메뉴 선택 : '
WRONG_IN_MSG = '잘못된 입력'
WRONG_NUM_MSG = '잘못된 번호 입력'
NAME_IN_MSG = '이름 : '
NEW_NAME_IN_MSG = '새 이름 : '
EXIST_NAME_MSG = '이미 있는 이름'
NOT_EXIST_NAME_MSG = '해당 이름은 없음'
```

문자열 상수만 추려 적당히 깔끔해 보일 정도로 이름을 지어보았다.
입력(input) 메시지에는 줄여서 IN, 출력(output) 메시지에는 줄여서 OUT을 붙여 구분해도 좋다.
어떤 방식이든 좋다.
자기 나름대로 일정한 규칙을 정해서 정리하기만 하면 된다.
나중에 작업하기 좋도록 등호(=)의 위치를 맞춰주는 것도 괜찮다.
아는 개발자 동생에게 배운 팁인데, 파일명을 const.py로 해서 상수를 관리하고 import 해서 사용하면 const.변수명의 형식으로 상수를 사용할 수 있어 상수와 변수 구분이 보다 명확해진다.

이제 지금까지 수정한 내용을 합쳐보자.

```
""" CONST """
NEW_LINE_MSG = '\n'
SEP_STAR_MSG = '*'
OPTION_1_MSG = '1. 이름 추가'
OPTION_2_MSG = '2. 이름 삭제'
OPTION_3_MSG = '3. 이름 수정'
OPTION_4_MSG = '4. 종료'
SELECT_O_MSG = '메뉴 선택 : '
WRONG_IN_MSG = '잘못된 입력'
WRONG_NUM_MSG = '잘못된 번호 입력'
NAME_IN_MSG = '이름 : '
NEW_NAME_IN_MSG = '새 이름 : '
EXIST_NAME_MSG = '이미 있는 이름'
NOT_EXIST_NAME_MSG = '해당 이름은 없음'

""" MAIN """
mylist = []

def menu():
    print(NEW_LINE_MSG
          + SEP_STAR_MSG * 19
          + NEW_LINE_MSG + OPTION_1_MSG
          + NEW_LINE_MSG + OPTION_2_MSG
          + NEW_LINE_MSG + OPTION_3_MSG
          + NEW_LINE_MSG + OPTION_4_MSG)
    n = input(SELECT_O_MSG)
    if not n.isdigit():
        print(WRONG_IN_MSG)
        menu()
    else:
        n = int(n)
        if n == 1:
            name = input(NAME_IN_MSG)
            if name not in mylist:
                mylist.append(name)
                print(mylist)
            else:
                print(EXIST_NAME_MSG)
        elif 1 < n < 4:
            name = input(NAME_IN_MSG)
            if name not in mylist:
                print(NOT_EXIST_NAME_MSG)
            else:
                if n == 2:
                    mylist.remove(name)
                else:
                    new_name = input(NEW_NAME_IN_MSG)
                    i = mylist.index(name)
                    mylist[i] = new_name
                print(mylist)
        elif n == 4:
            return
        else:
            print(WRONG_NUM_MSG)
        menu()

menu()
```

간단한 코드를 바꾼 것이라 처음보다 오히려 복잡해진 것 같기도 하지만, 뭔가 신경 써서 작성한 코드라는 티는 팍팍 난다.

계속 강조하지만, 일단 구현 실력을 갖추는 것이 우선이다.
하지만 어느 정도 실력이 갖춰지고 나서는, 좋은 형식의 코드를 작성하는 연습 또한 병행되어야 한다고 생각한다.
실력이 늘수록 어려운 프로젝트를 진행할 수 있게 되고, 어려운 프로젝트를 진행하게 되면 협업의 필요성이 높아진다.
협업에서 중요한 것은 협업자와의 소통이고, 소통에 필요한 것은 언어, 개발자의 언어는 프로그래밍 언어다.
다른 사람이 알아보기 좋은 코드를 작성하는 것은 조리 있게 말할 수 있는 언변을 가진 것과 같다고 생각한다.
코드만 봐도 작성자의 실력을 가늠할 수 있다는 말은 코드의 이러한 특성 때문에 나온 말일 것이다.

지금까지 내가 생각하는 좋은 코드 작성법에 대해 정리해 보았다.
미흡한 글이지만, 누군가에게 도움이 될 수 있었으면 한다.