week1

context

1. 영상 강의 학습
2. 과제 및 퀴즈 제출
3. 과제 코드 리뷰
4. 기타 활동

블로그 : https://velog.io/@changyong93

github : https://github.com/changyong93/boost_camp_ai_tech

---

1. 영상 강의

- Python Basic for AI : 완강 및 정리 완료
- AI Math : 6강까지 학습 완료(금일 및 주말을 이용해서 완강 예정)

---

2. 과제 및 퀴즈 제출

- 필수 과제 1~5 제출 완료
- AI Math 1~10강 퀴즈 제출 완료
- 선택과제는 주말에 풀어볼 예정

---

3. 과제 코드 리뷰

- 결론부터 얘기하자면, 우선 main()함수를 제외한 함수를 구현하는데에는 큰 문제는 없었지만
  무리한 list comprehension을 통해 가독성이 조금 떨어진다는 의견이 많았음
  ⇒ 코드 성능도 중요하지만 어느 정도 가독성도 확보해야 할 것(2주차 과제부터 적용)
- main() 함수를 구현 시 문제 조건을 모두 적용을 잘 했지만, print문으로 출력하는 text 내용이 문제가 될지 몰랐어서 생각보다 4~5번 과제를 푸는데 오랜 시간이 걸림
- 또한, main()함수 역시 어느 정도 기능적으로 구현을 했지만 가독성이 떨어진다는 평가가 많았음으로 이 또한 2주차 부터 개선할 예정

[필수과제 1] Basic Math

- get_mean() 이나 get_median() 함수 구현 시 입력된 값을 array로 변환하고, 그 값을 np.mean()을 활용해서 구했는데, array로 변경할 필요가 없었음을 알게 되었음.
- 또한 강의를 통해 알게된 점은, python의 max() 함수보다 numpy의 max가 함수 내부 언어의 차이로 속도가 더 빠름을 알게 됨 ⇒ 다음부터는 list의 경우 max()함수를 사용하지 말 것

```
def get_max(number_list):
    return max(number_list) #X => np.max(number_list)

def get_min(number_list):
    return min(number_list) #X => np.min(number_list)

def get_mean(number_list):
    return np.mean(np.array(number_list)) #X => np.mean(number_list)
```

```
def get_median(number_list):
    return np.median(np.array(number_list)) #X => np.median(number_list)
```

[필수과제 2] Text Processing 1

- re 함수를 통해 정규표현식으로 전처리
- 지적사항 없음

```
import re
def normalize(input_string):
    normalized_string = re.sub(" +"," ", input_string.lower()).strip()
    return normalized_string


def no_vowels(input_string):
    no_vowel_string = re.sub("[aeiouAEIOU]","",input_string)
    return no_vowel_string
```

[필수과제 3] Text Processing 2

- 정규표현식, dict, str method를 이용한 문제 해결
- digits_to_words() 문제를 풀 때 나의 경우는 digits과 strings을 key:value로 가지는 dictionary()로 만들어서 풀었으며, 팀원 중 한 분은 strings의 내용을 list + in 명령어를 통해 구현
- input_string에 여러 개의 데이터가 입력됐을 때 in 을 쓰면 시간 복잡도 O(n)이고, dict로 구현하면 O(1)이기에, 개인적으로 dict가 적합하다고 판단

```
import re
def digits_to_words(input_string):
    digits = "0123456789"
    strings = "zero one two three four five six seven eight nine"
    digit_to_string = {d : s for d,s in zip(list(digits), strings.split())}
    digit_string = " ".join([digit_to_string[n] for n in re.sub("[^\d]","",input_string)])
    return digit_string

def to_camel_case(underscore_str):
    camelcase_str = "".join([s.capitalize() if i != 0 else s.lower() for i, s in enumerate(re.sub("(_)+"," ",underscore_str).split())]
    return underscore_str if underscore_str == "" or "_" not in underscore_str else camelcase_str
```

[필수과제 4] BaseBall

- main()에서 내용을 출력해주는 print()에 대한 오류 확인에 시간이 오래 소요됐었음
- 각 함수를 구현하며 조원들이 작성한 코드를 통해 많은 것을 배울 수 있었음

```
def is_digit(user_input_number):
    try:
      check_integer = int(user_input_number)
      result = True
    except ValueError:
      result = False
    return result
```

- 내 경우는 정수를 확인하기 위해 try except을 사용했지만 isdigit() 함수를 사용하면 문자열에 숫자만 있을 경우 True를 반환, 나머지는 False를 반환하는 것을 알게 됨

```
def is_between_100_and_999(user_input_number)
    integer = int(user_input_number)
    result = True if 100 <= integer and integer < 1000 else False
    return result
```

- 모든 팀원이 비슷하게 코드를 작성함
- len(user_input_number) == 3으로 하면 깔끔하게 작성됨을 확인!!

```
def is_duplicated_number(three_digit):
    Count_string_digit = Counter(three_digit)
    result = True if len(Count_string_digit) < 3 else False
    return result

def is_duplicated_number(three_digit):
    result = True if len(set(three_digit)) < 3 else False
    return result
```

- 내 경우, Counter 함수를 통해 숫자로 된 문자열의 개수를 세고, 그 key 값이 3개 이하면 False, 아니면 True를 반환
- 더 나은 코드라 생각되는 코드는 아래 코드로 집합인 set을 활용한 코드

```
def is_validated_number(user_input_number):
    result = False
    if is_digit(user_input_number):
      check_input_string = lambda x : all([is_between_100_and_999(x),not is_duplicated_number(x)])
      result = check_input_string(user_input_number)
    # ==================================
    return result
```

- 팀원 별로 거의 코드가 비슷하지만 조금씩만 다르게 사용
- 내 경우는 숫자가 아니면 바로 False를 리턴하고, True일 경우
  - 100~999사이 및 중복값이 없으면 True를 반환하는 lambda 함수를 구현하여 결과 출력

```
def get_not_duplicated_three_digit_number():
check = False
    while not(check):
      result = get_random_number()
      check = True if len(Counter(str(result))) == 3 else False
    # ==================================
    return result
```

- is_duplicated_number(three_digit): 문제와 마찬가지로 Counter로 값을 판단하여 True일 경우 결과 return
- 이거 역시 Set을 활용했으면 코드가 더 간결했을 거라 판단함

```
def get_strikes_or_ball(user_input_number, random_number):
    result = [0,0]
    for i, n in enumerate(user_input_number):
      if n == random_number[i]:
        result[0] += 1
      elif n in random_number:
        result[1] += 1
    return result
```

- strikes와 ball 판단하는 문제로, 이 문제에 한해선 내 코드가 가장 깔끔한 코드라는 얘기를 해주심

```
def is_yes(one_more_input):
    result = True if one_more_input.lower() in ['y',"yes"] else False
    # ==================================
    return result


def is_no(one_more_input):
    result = True if one_more_input.lower() in ["n", "no"] else False
    return result
```

- is_no or is_yes 역시 내 코드가 가장 깔끔하고 이해하기 쉽다는 얘기를 해주심

```
def main():
    print("Play Baseball")

    user_input = 999
    get_new_random_number = True
    while get_new_random_number:
      random_number = str(get_not_duplicated_three_digit_number())
      print("Random Number is : ", random_number)
      restart = True

      while restart:
        user_input = input('Input guess number(100~999): ')
        if user_input == '0':
          restart = False
          get_new_random_number = False
          continue

        if not is_validated_number(user_input):
          print("Wrong Input")
          continue

        strikes_balls = get_strikes_or_ball(user_input,random_number)
        print(f"{user_input} {random_number} {str(strikes_balls[0])} STRIKES, {str(strikes_balls[1])} BALLS")

        if strikes_balls[0] != 3:
          continue

        one_more_game = input("one more game?: ")
        while is_yes(one_more_game)== False and is_no(one_more_game)==False and one_more_game != '0':
          print("Wrong Input")
          one_more_game = input("one more game?: ")

        if is_yes(one_more_game) != True:
          get_new_random_number = False
        restart = False
    # ==================================
    print("Thank you for using this program")
    print("End of the Game")
```

- main 함수를 구현하기 가장 힘들었는데 코드 순서로는
  1. get_new_random_number ==True면 진행, False면 게임 종료
  2. random_number 값을 받아오고 게임이 진행되는데
     a. 만약 값이 0이면 프로그램 종료
     b. 값이 잘못됐으면 if - continue를 통해 input_number를 다시 받아옴
     c. 값이 잘 입력됐으면 진행
  3. strikes 와 ball의 빈도수를 세는데
     a. strikes가 3미만이면 2번과정 재진행
     b. strikes = 3일 경우 입력을 다시 받는데
        i. ['no', 'n', 0] 이면 게임 종료
        ii. ['y','yes']면 2번 과정 재진행

- 결론적으로 코드는 괜찮게 구현했으나, continue말고 elif를 사용해서 가독성을 높였으면 좋겠다는 의견이 있었음

[필수과제 5] MorseCode

```
import re
def is_help_command(user_input):
    result = True if user_input.upper() in ["H", "HELP"] else False
    return result
```

- user_input 이 Help, H면 True를 반환하는 것인데, 굳이 if else로 True False 쓸 필요는 없을 것으로 보임

```
def is_validated_english_sentence(user_input):
    if re.findall("[\d]",user_input):
      return False
    if re.findall("[_@#$%^&*()\-+=\[\]{}\"';:\\\|`~]",user_input):
      return False
    if not re.sub("[,.!? ]","",user_input):
      return False
    return True
```

- 해당 문제를 풀며 특수문자가 포함되면 False를 반환했는데,

```
def is_validated_morse_code(user_input):
    if re.sub("[.\-, ]", "", user_input):
      return False
    if any(morse not in get_morse_code_dict().values() for morse in user_input.split()):
      return False
    return True
```

- 모스부호를 만드는데 필요한 . \ _ 빈칸을 제외할 경우 값이 남아있으면 False
- input을 get_morse_code_dict().values()에 포함이 되지 않은게 하나라도 없으면 False 반환

```
def get_cleaned_english_sentence(raw_english_sentence):
    result = re.sub("[.,!?]","",raw_english_sentence).strip()
    return result
```

- 정규표현식으로 구현

```
def encoding_character(english_character):
    morse_code_dict = get_morse_code_dict()
    return morse_code_dict[english_character]
```

- 간단하니 생략

```
def decoding_sentence(morse_sentence):
    result = ' '.join(['',join([decoding_character(alphabet_morse) for alphabet_morse in word_morse.split()]) for word_morse in morse_
    return result
```

- 해당 코드는 list comprehension으로 했지만, 가독성이 떨어진다는 의견을 들음
- 다음번엔 조금 느리더라도 적절히 for문과 섞어서 진행해볼 예정

```
def encoding_sentence(english_sentence):
    cleaned_english_sentence = get_cleaned_english_sentence(english_sentence).upper()
    result = '  '.join([' '.join([encoding_character(alphabet.upper()) for alphabet in word]) for word in cleaned_english_sentence.spl
    return result
```

-해당 코드 역시 decoding_sentence()함수와 동일한 의견을 들음

```
def main():
    print("Morse Code Program!!")
    # ===Modify codes below=============
    run_program = True

    while run_program:
      message = input("Input your massage(H - Help, 0 - Exit): ")

      while not is_validated_english_sentence(message) and not is_validated_morse_code(message) and not message.upper() in ["0","H","H
        print("Wrong Input")
        message = input("Input your massage(H - Help, 0 - Exit): ")

      if is_validated_morse_code(message):
        print(decoding_sentence(message))
        continue

      if message == '0':
        run_program = False
      elif message.upper() in ['H',"HELP"]:
        print(get_help_message())
      else:
        print(encoding_sentence(message))
    # ================================
    print("Good Bye")
    print("Morse Code Program Finished!!")

if __name__ == "__main__":
    main()
```

- 해당 코드를 팀원들과 비교한 결과
  - 불필요한 코드도 많았고, 가독성도 떨어졌으며, 함수에 대한 설명이 있으면 좋겠다는 의견을 받음

기타 활동

- 강의 영상 블로그 정리
  - 겉핥기 식으로 알던 내용에 대해 조금 더 딥하게 접근하고 있어서 시간이 많이 소요되는중..
  - 7~10강 은 주말을 이용해 정리할 예정
  -

- 알고리즘 문제 풀기
  - 일당 2개씩 모더레이터가 선정하여 풀기로 하였으나, 첫주라 아직 적응이 안됐고, 조금 어수선해서 아직 풀진 못함
  - 이 문제(총 10개) 역시 주말에 풀 예정

- 피어세션
  - 조금씩 정리되는 느낌이 있으며, 다음주부터 조금 더 적극적으로 토론하고 자료를 공유하여 팀원 모두가 win-win할 수 있게 진행해볼 예정