CS 1: Introduction to Computer Programming

Recitation 4: Exceptions and Dictionaries

In today's syntax recitation, we'll cover exceptions and dictionaries!

Problems

Anagrams (topics: Dictionaries)

Lets find all of the anagrams for a given word. An anagram is a rearrangement of the letters in a word that still yield a valid word. Assume that we have predefined a WORDS set, which contains all the possible valid words in the english language. Implement both `build_words_map` and `anagrams`.

Hint: Sorted returns a list of all the letters in a word, sorted by alphabetical order.

```
1
2  WORDS: set[str] = {...} #Assume that words is defined
3
4  def build_words_map(words: set[str]) -> dict[str, set[str]]:
5      """
6      Maps a given sequence of sorted letters to all the possible combinations of those letters that are
           considered valid English words.
7
8      For example, if we have the valid english words {"cat", "act", "tar", "rat", "art"}, we should
           return the resulting dictionary:
9
10     {"act": {"act", "cat"}, "art": {"art", "rat", "tar"}}
11     """
12     words_map = {}
13
14     for word in words:
15         key = ''.join(sorted(word)) #Returns the sorted version of the word
16         if                                                  :
17                                                             = set()
18
19     return words_map
20
21 WORDS_MAP = build_words_map(WORDS)
22
23 def anagrams(word: str) -> list[str]:
24     """
25     Returns a list of anagrams for a given letters. If there are no valid words that this word maps to,
           then return an empty list.
26     """
27     key = ''.join(sorted(word))
28
29     if                                                  :
30         return
31
32     return []
```

Grading Script (topics: Dictionaries and exceptions)

Lets write a grading script together!

Implement `convert_grade`, which takes in a string representing the grade of a student, and returns the integer representation of that string. However, if the input cannot be turned into an integer or if the input is less than 0 or greater than 100, raise a ValueError.

```
1  def convert_grade(grade: str) -> int:
2      [          ]:
3          grade_int = int(grade)
4      [                              ]:
5          [          ] ValueError("Grade must be a number.")
6
7      if [                              ]:
8          [          ] ValueError("Grade must be between 0 and 100.")
9      return grade_int
```

Below is some code to process grades. If `convert_grade` raises a `ValueError`, then we print the error message associated with that.

```
10 grades = {}
11 done = False
12 while not done:
13     try:
14         name = input(Enter the name of the student: ")
15         grade = input(Enter your grade: )
16         numeric_grade = convert_grade(grade)
17         if name in grades:
18             grades[name].append(numeric_grade)
19         else:
20             grades[name] = [numeric_grade]
21         done = bool(input(Are you done? (True/False)))
22     except ValueError as e:
23         print(e)
```

Now that grades have been finalized, use `grades` to compute the highest average score across all students.

```
24 best_student = [                ]
25 best_score = [                ]
26 for entry in grades:
27     avg = sum([                ]) / [                ]
28     if [                              ]:
29         best_student = [                ]
30         best_score = [                ]
31 print((best_student, best_score))
```