CS 1: Introduction to Computer Programming

Recitation 4: Exceptions and Dictionaries Solutions

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

Problem Solutions

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.

```python

WORDS: set[str] = {...} #Assume that words is defined

def build_words_map(words: set[str]) -> dict[str, set[str]]:
    """
    Maps a given sequence of sorted letters to all the possible combinations of those letters that are considered valid English words.

    For example, if we have the valid english words {"cat", "act", "tar", "rat", "art"}, we should return the resulting dictionary:

    {"act": {"act", "cat"}, "art": {"art", "rat", "tar"}}
    """
    words_map = {}

    for word in words:
        key = ''.join(sorted(word)) #Returns the sorted version of the word
        if key not in words_map:
            words_map[key] = set()
        words_map[key].add(word)
    return words_map

WORDS_MAP = build_words_map(WORDS)

def anagrams(word: str) -> list[str]:
    """
    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.
    """
    key = ''.join(sorted(word))

    if key in WORDS_MAP:
        return WORDS_MAP[key]

    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.

```
def convert_grade(grade: str) -> int:
    try :
        grade_int = int(grade)
    except ValueError :
        raise  ValueError("Grade must be a number.")

    if grade_int < 0 or grade_int > 100 :
        raise)  ValueError("Grade must be between 0 and 100.")
    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.

```
grades = {}
done = False
while not done:
    try:
        name = input(Enter the name of the student: ")
        grade = input(Enter your grade: )
        numeric_grade = convert_grade(grade)
        if name in grades:
            grades[name].append(numeric_grade)
        else:
            grades[name] = [numeric_grade]
        done = bool(input(Are you done? (True/False)))
    except ValueError as e:
        print(e)
```

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

```
best_student = None
best_score = -1
for entry in grades:
    avg = sum( grades[entry] ) / len(grades[entry]
    if avg > best_score :
        best_student = entry
        best_score = avg
print((best_student, best_score))
```