Python Practice Worksheet 3

Focus: Sets, Tuples, Dictionaries, Strings & Comprehensions!

# SETS: Discovering Uniqueness

1. Open your Python editor and type any sentence. For example:
2. sentence = "Learning Python is exciting!"
3. Read your sentence and pick out all the vowels. For reference: 'a', 'e', 'i', 'o', 'u'
4. Create a set containing all unique vowels from your sentence using set comprehension.
5. vowels\_found = {ch.lower() for ch in sentence if ch.lower() in "aeiou"}
6. Add the letter 'y' to your set.
7. vowels\_found.add('y')
8. Print your resulting set to see the unique vowels.
9. print(vowels\_found)
10. Remove a vowel, like 'e', from your set.
11. vowels\_found.remove('e')
12. Reflect: Try printing the set after each operation to see the changes.

Question: What property of sets ensures each vowel appears only once?

# TUPLES: Packed with Meaning

1. Think of a book you like. Example:
2. book = ("The Hobbit", "J.R.R. Tolkien", 1937)
3. Create your tuple with (title, author, year).
4. Unpack it:
5. title, author, year = book
6. Print the information using f-strings:
7. print(f"'{title}' by {author}, published in {year}")
8. Try again for another book:
9. book2 = ("Dune", "Frank Herbert", 1965)t, a, y = book2print(f"'{t}' by {a}, published in {y}")
10. Swap the order:
11. swapped = (year, author, title)print(swapped)
12. Reflect: Try unpacking swapped and observe the results.

Question: How does tuple unpacking make your code more readable and fun to write?

# DICTIONARIES: Count the Words, Rule the Data

1. Write or copy a short paragraph.
2. text = "Python programming is fun. Python is powerful."
3. Split it into words:
4. words = text.split()
5. Initialize a dictionary:
6. freq = {}
7. Count frequencies:
8. for word in words:
9. freq[word] = freq.get(word, 0) + 1
10. Print the dictionary:
11. print(freq)
12. Sort and print the most common word:
13. most\_common = max(freq, key=freq.get)
14. print(most\_common, freq[most\_common])
15. Remove a word:
16. del freq["Python"]
17. Reflect: Try printing after each step to watch the changes.

Question: Why is a dictionary perfect for keeping track of word counts?

# ADVANCED STRINGS: Transform and Analyze

1. Type a favorite phrase.
2. sentence = "Hello world!"
3. Replace spaces with underscores:
4. s = sentence.replace(" ", "\_")
5. Convert to uppercase:
6. s\_upper = s.upper()
7. Check if it starts with a vowel:
8. print("Starts with vowel:", s\_upper[0] in "AEIOU")
9. Print each transformation:
10. print(s)print(s\_upper)
11. Challenge: Use startswith() to check for "HEL".
12. print(s\_upper.startswith("HEL"))
13. Reflect: Experiment with more string methods like .count() or .find().

Question: Which string methods did you use, and what does each one accomplish?

# LIST COMPREHENSION: Lightning-Fast Logic

1. Write a loop for squares of even numbers:
2. squares = []
3. for n in range(1, 21):
4. if n % 2 == 0:
5. squares.append(n\*\*2)
6. Rewrite as a list comprehension:
7. squares = [n\*\*2 for n in range(1, 21) if n % 2 == 0]
8. Print your list:
9. print(squares)
10. Modify for numbers divisible by 4:
11. quads = [n for n in range(1, 21) if n % 4 == 0]
12. Try cubes of odd numbers:
13. cube\_odds = [n\*\*3 for n in range(1, 21) if n % 2 == 1]
14. Reflect: Compare output of all three lists.

Question: How does a list comprehension make your code more concise and readable?

# SET COMPREHENSION: Unique and Efficient

1. Type a string.
2. s = "Elegant examples excite everyone."
3. Create a set of unique lowercase letters:
4. letters = {ch for ch in s.lower() if ch.isalpha()}
5. Print your set:
6. print(letters)
7. Filter only consonants:
8. consonants = {ch for ch in letters if ch not in 'aeiou'}
9. Reflect: Print both sets and notice the difference.

Question: Why is set comprehension an efficient way to gather unique items?

# DICTIONARY COMPREHENSION: Map It Out!

1. Take "PYTHONIC" and map each character:
2. d = {ch: ord(ch) for ch in "PYTHONIC"}
3. Print the dictionary:
4. print(d)
5. Try mapping lowercase letters:
6. d\_lower = {ch: ord(ch) for ch in "pythonic"}
7. Experiment: Map numbers to their squares:
8. squares = {n: n\*\*2 for n in range(5)}
9. Reflect: Compare and discuss results.

Question: What does the ord() function do, and why is dictionary comprehension useful here?

# TUPLES IN LISTS: Find the Champion

1. Create a list of tuples:
2. scores = [("Alice", 92), ("Bob", 85), ("Chris", 97)]
3. Find the highest score:
4. best = max(scores, key=lambda x: x[1])print(best[0], best[1])
5. Find the lowest score:
6. worst = min(scores, key=lambda x: x[1])print(worst[0], worst[1])
7. Reflect: Try with different data and see how results change.

Question: How does using max() with a key argument simplify finding the winner?

# MERGING DICTIONARIES: Combine & Conquer

1. Create two dictionaries:
2. d1 = {'x': 1, 'y': 2}d2 = {'y': 3, 'z': 4}
3. Merge them:
4. d = {\*\*d1, \*\*d2}
5. Update a value:
6. d['x'] = 42
7. Print your dictionary:
8. print(d)
9. Add a new value:
10. d['w'] = 99
11. Reflect: See what happens to overlapping keys ('y').

Question: What happens when both dictionaries have the same key during merging?

# STRING FORMATTING: Names in Style

1. Input names:
2. first = "Taylor"
3. last = "Swift"
4. Swap and join using f-string:
5. print(f"{last}-{first}")
6. Print with extra info:
7. age = 30print(f"{last}-{first}, Age: {age}")
8. Try formatting favorite color:
9. color = "blue"print(f"{last}, {first}: Favorite color is {color}")
10. Utilize the “.format()” method to organize a string that contains variable
    1. Ex:“{0} is {1} favorite {2}.format(x, y, z)!”
11. Reflect: Experiment with more formats.

Question: What advantages do f-strings offer compared to older formatting methods?

# Challenges

1. Keyword Search Tool: Write a program that reads a paragraph from the student, splits it into words, and allows them to search for a keyword. The program should report its frequency, highlight every appearance, and print sentences containing it.
2. # Try: Use .count(), .replace(), and .split() methods
3. Student Score Tracker: Use a dictionary to map student names to tuples of quiz scores. Write functions to add, update, and calculate the top and bottom averages.
4. # Try: Use dictionary and tuple operations, plus list comprehensions
5. Custom Text Formatter: Transform a sentence: replace spaces, capitalize vowels, count unique consonants.
6. # Try: Use string methods, set comprehensions, and formatting
7. Data Summary Dashboard: Given a list of dictionaries for products, display total inventory value, most expensive item, and sets of unique names and initials.
8. # Try: Use for-loops, comprehensions, and dictionary access
9. Puzzle Generator: Create a list of tuples pairing words with their lengths, shuffle, and challenge a student to match them.
10. # Try: Use tuples, lists, and the random library

Choose one or more challenges and combine your new skills to create something cool!

# Reflection

1. Which activity did you find most exciting or challenging? Why?
2. What new skills or concepts stood out to you as especially useful?
3. How could you use these Python techniques in your own creative projects?