Lab 06: Default arguments, args and kwargs

Total Points: 10 (Correctness) + 1 extra credit (optional, early submission) + 2 extra credits (optional, additional problem).
Due: Tuesday, February 20th, 11:59 pm (SD time)

Starter Files

Download lab06.zip. Inside the archive, you will find starter files for the questions in this lab. You can't import anything to solve the problems.

Extra Credits Opportunity

You have the opportunity to receive a

- **1 point** extra credit on this lab if you submit your last attempt early (refer to this section of each lab for the early submission deadline). Note: Each lab is graded out of 10 points, and you could possibly have more than 10 points in one lab.
- 2 points extra credit on this lab if you solve the last question.

Early Submission Date (lab06): Friday, February 16th, 11:59pm (SD time)

Testing

After finishing each question, run the file with the following command-line options to avoid compile-time errors and test the correctness of your implementation:

- No options: `>>> python3 lab06.py`
- Interactive mode: `>>> python3 -i lab06.py`
- Doctest **(Recommended)**: `>>> python3 -m doctest lab06.py`

For Windows users, please use `py` or `python` instead of `python3`.

Important Requirements and Notes

1. **DO NOT IMPORT ANY PACKAGES.**
2. It may be useful to add doctests when the given doctests do not cover all cases to avoid losing points from the Autograder/any hidden tests, but you are not required to add doctests in the labs.

3. Style/asserts are not required or graded on any labs, but it's recommended that you also follow the style guide for clean code. Method descriptions are provided in this lab.
4. You may assume all inputs are valid.

Required Questions

Unfortunately, the Valentine's day is over, and you need to return to exploring different areas of Pythonia while learning new skills. This time, you were heading to `ArgKwanda`, the land of *args and kwargs*. On your way to this land you met a tutor, who suggested that you borrow the ancient ***Enchanted Spellbook*** to make this journey more exciting.

Question 1:

You really liked the idea that you could become an aspiring Spellcaster! In this magical adventure, you'll be summoned to craft Python spells using the ancient Enchanted Spellbook.

Question 1.1:

You, the aspiring Spellcaster, are summoned to craft Python spells using the ancient Enchanted Spellbook. Your first challenge is to create the "*Flamestrike Spell.*" Write a function that takes an *unknown* number of nonnegative integers as arguments, which represent the intensity of the flames, and returns a list containing the damage dealt by unleashing the flames of the Flamestrike Spell. The function to calculate the damage is: $d(x) = 10x^2$

Requirement:

No explicit for/while loops. You may choose between list comprehensions or map/filter/lambda function.

```
def flamestrike_spell(*args):
    """
    >>> flamestrike_spell(1, 2, 3, 4, 5)
```

```
    [10, 40, 90, 160, 250]
    >>> flamestrike_spell(0, 1, 2, 3)
    [0, 10, 40, 90]
    >>> flamestrike_spell(78, 142)
    [60840, 201640]
    """
```

Question 1.2:

As your Spellcaster journey continues, you acquire the "*Levitation Incantation*." However, as a novice in the realm of spellcasting, your abilities have limitations, and you cannot levitate just any object. Instead, there is a specific weight threshold that determines what you can and cannot levitate.

Write a function that takes a list of nonnegative integers, which represent the weights of each object, and a weight threshold (nonnegative integer). You can levitate all objects below or at the weight threshold (nonnegative integer). The function should return a list of weights that can be levitated.

Requirement:

No explicit for/while loops. You may choose between list/dictionary comprehensions or map/filter/lambdas

```
def levitation_incantation(weights, weight_threshold = 10):
    """
    >>> levitation_incantation([5, 8, 12, 15, 18])
    [5, 8]
    >>> levitation_incantation([10, 20, 30, 40, 50], 35)
    [10, 20, 30]
    >>> levitation_incantation([22, 33, 44], weight_threshold = 44)
    [22, 33, 44]
    """
```

Question 1.3:

You are responsible for a new task – to count the number of **enchanted spells** in the ancient Enchanted Spellbook. Each spell in the spellbook is represented as a string, and an enchanted spell is one that contains the word "Enchanted" (case-sensitive) within it. However, you have learned that not all spells are equally enchanting. You now have the ability to set a minimum length for a spell to be considered *enchanting*.

Write a function that takes an unknown number of spell names (strings) and a length threshold (nonnegative integer). The function should return the total number of enchanted spells in the input list of spells that meet both the length threshold and the presence of "Enchanted."

Requirement:

> **No explicit for/while loops**. You may choose between list comprehensions or map/filter.

```
def enchanted_spell_counter(*args, length=15):
    """
    >>> enchanted_spell_counter("Enchanted Fireball Spell",
"Levitation Incantation", "Enchanted Charm", "Teleportation Spell")
    2
    >>> enchanted_spell_counter("Mystic Incantation", "Enchanted
Spellbook", "Crystal Ball Gaze", length=20)
    0
    >>> enchanted_spell_counter("Enchanted Spellbook", "Mystic
Incantation")
    1
    """
```

Question 2:

After learning all sorts of different spells, you and fellow spellcasters are eager to showcase your enchanting abilities.

Write a function called `spell_showcase` that calculates the number of **enchanted spells** for each spellcaster. The function takes a `length_threshold` (as an integer) and keyword arguments where each keyword is the spellcaster's name and the corresponding value is the list of spells (as strings) that the spellcaster knows. The function should return a dictionary where the keys are spellcaster names, and values are the number of enchanted spells each spellcaster has mastered.

Requirement:

No explicit for/while loops. You may choose between list/dictionary comprehensions or map/filter.

Hints:

- You may want to call `enchanted_spell_counter()` above.
- Asterisk operator (*) helps you `unpack` a list into function arguments:

```
def add(a, b, c):
    """
    >>> numbers = [1, 2, 3]
    >>> add(*numbers)
        6
    """
    return a + b + c
```

```
def spell_showcase(length_threshold, **kwargs):
    """
    >>> spell_showcase(5, hermione=['Enchanted Charm', 'Teleportation Spell'], dumbledore=['Mystic Incantation'])
    {'hermione': 1, 'dumbledore': 0}
    >>> spell_showcase(10, harry=['Wingardium Leviosa', 'Expelliarmus'], voldemort=['Avada Kedavra'])
    {'harry': 0, 'voldemort': 0}
    >>> spell_showcase(20, gandalf=['Enchanted Spellbook', 'Crystal Ball Gaze', 'Enchanted Teleportation Spell'], saruman=['Mystic Incantation'])
    {'gandalf': 1, 'saruman': 0}
    """
```

Question 3:

In Pythonia, each spellcaster has a certain level of experience and wants to level up their spell-casting abilities. You want to use your coding skills to help fellow spellcasters make a level-up system.

Write a function called `level_up` that calculates the new spell-casting levels based on their current experience (*args), additional experience (**kwargs), and a leveling factor (int). If the answer is a float, just remove the decimal part.

- `leveling_factor` (positive integer) is the factor by which each spellcaster's experience is divided to determine the number of levels they can level up.

- `*args` (tuple) represents the spellcasters, each tuple contains the spellcaster's name (string) and their current experience level (integer).
- `**kwargs` represents the additional experience for each spellcaster, where the keyword is the spellcaster's name (string), and the value is their additional experience level (integer).

Note: you can assume there to be no name duplicates within the tuples.

Requirement: Dictionary Comprehension only. No explicit for/while loops. No helper functions. No map/filter.

For example,

Input	Output	Reason
`factor = 10` `level_up(factor, ("merlin", 350), ("hermione", 500), merlin = 100, hermione = 50)`	`{'merlin': 45, 'hermione': 55}`	1. Merlin: 350 (current experience) + 100 (additional experience) = 450. 450//10 (leveling factor) = 45 2. `Hermione`: 500 (current experience) + 50 (additional experience) = 550 550//10 (leveling factor) = 55.

```
def level_up(leveling_factor, *args, **kwargs):
    """
    >>> factor = 10
    >>> level_up(factor, ("merlin", 350), ("hermione", 500), merlin =
100, hermione = 50)
    {'merlin': 45, 'hermione': 55}

    >>> factor = 50
    >>> level_up(factor, ("merlin", 350), ("gandalf", 500),
("hermione", 250), ("dumbledore", 700), merlin = 100, hermione = 50)
    {'merlin': 9, 'gandalf': 10, 'hermione': 6, 'dumbledore': 14}

    >>> factor = 30
    >>> level_up(factor, ("harry", 200), ("ron", 150), ("hermione",
250), ("neville", 175), harry = 50, neville = 30)
    {'harry': 8, 'ron': 5, 'hermione': 8, 'neville': 6}
    """
```

Question 4:

On the way to returning the ancient Enchanting Book, you run into some magical creatures trapped in cages. You have decided to use your Python spells to set the creatures free. However, the cages are enchanted with a magical lock, and your spells need to surpass a certain level of power to break them open.

Write a higher-order function `open_cages` that takes in an unknown number of (spell name, magical strength) keyword pairs in the form of `spell_name = strength`, where `spell_name` is a string and `strength` is an integer (there will always be at least 1 pair). It calculates the total magic potency according to the given formula below.

Then `open_cages` returns a **function** that takes in a threshold value (as a positive number) and compares it with the calculated magic potency. The returned function should return based on the provided rules.

Details:

- Calculate the total magic potency using the formula below:

$$p = \sum_{i=1}^{n} ((strength)^2 * len(spell\_name))$$

  In other words, calculate the square of `strength`, then multiply by `len(spell_name)` for each spell and then find the sum.

- Then compare the total magic potency, p, against a positive threshold (either integer or float)
  - If p > threshold
    - Return the spell name with the **highest** strength and the string "Good Job!" as a tuple.
  - If p < threshold
    - Return the spell name with the **lowest** strength and the string "Try Again!" as a tuple.
  - If p == threshold
    - The cages cannot be opened yet. Return the string "One more round!"

In the event of multiple spells having the highest/lowest strength, return the spell name that occurs first in the input.

Hint: use built-in function max that takes an optional parameter (how to search for the maximum).

Example:
Input: 'Levitation_Incantation' = 4, 'Enchanted_Charm' = 10

Steps:

1. Compute the potency of each spell
    a. 'Levitation_Incantation': ((4**2) * 22)
    b. 'Enchanted_Charm': ((10**2) * 15)
2. Compute the total potency:
    a. 352+1500 = 1852
3. Then a threshold of 1000 is given: Compare 1852 with the threshold:
    a. 1852 > 1000
4. **Return** ('Enchanted_Charm', 'Good Job')

Output:

5. ('Enchanted_Charm', 'Good Job')

Requirement:

No explicit for/while loops. You may choose between list comprehensions or map/filter.

```
def open_cages(**kwargs):
    """
    >>> magic = open_cages(Levitation_Incantation = 4, Enchanted_Charm
= 10)
    >>> magic(1000)
    ('Enchanted_Charm', 'Good Job!')
    >>> magic(6000)
    ('Levitation_Incantation', 'Try Again!')
    >>> magic(1852)
    'One more round!'
    """
```

Question 5:

You have successfully saved the magical creatures. However, you found out that they belonged to an evil ancient sorcerer. Now, You are on a quest to defeat the ancient sorcerer. However, your spells may not be powerful enough so you have decided to collect magical ingredients to strengthen your spells.

Each spell requires a specific set of magical ingredients, some of which you already possess, while others can be obtained from the magical store. To enhance your power, you have the option to obtain additional quantities of ingredients from the store.

Write a function `spell_resources` takes three types of parameters:

- **limit**: (An non-negative integer): represents the maximum number of times you want to cast the spell.
- ***args**: An arbitrary number of tuples, each in the format (`'ingredient_name'`, `amount`), where the `ingredient_name` is the magical ingredient's name (string), and the amount is the quantity you currently have (non-negative integer).
- ****kwargs:** Keyword arguments in the format `ingredient_name=amount`, where the `ingredient_name` is a string representing the magical ingredient's name, and amount is a positive integer representing the quantity *needed* to cast the spell **once**.

The function should return a **dictionary** with keys representing **ingredient names** for the ingredients you **don't** have enough of and values are the amount needed to buy from the magical store to meet the spell requirements considering the limit.

For example,

Input	**Output**	**Reason**
limit=2, ('unicorn_hair', 4), ('phoenix_feather', 2), 'unicorn_hair' = 3, 'dragon_scale' = 5	{'unicorn_hair': 2, 'dragon_scale': 10}	• Casting the spell once needs 3 'unicorn_hair' and 5 'dragon_scale' • You want to cast the spell twice • You have 4 'unicorn_hair' and 'phoenix_feather'

		• You need (3*2 - 4) ‘unicorn_hair’ and (5*2) ‘dragon_scale’ • **Return** {‘unicorn_hair’: 2, ‘dragon_scale’: 10}

```
def spell_resources(limit, *args
    """
    >>> spell_resources(2, ('unicorn_hair', 4), ('phoenix_feather',
2), unicorn_hair = 3, dragon_scale = 5)
    {'unicorn_hair': 2, 'dragon_scale': 10}
    >>> spell_resources(3, ('moonstone_dust', 0), ('starlight_petals',
2), moonstone_dust = 3, starlight_petals = 5)
    {'moonstone_dust': 9, 'starlight_petals': 13}
    """
```

Question 6 (Extra Credit):

Now you have powered up your Python spells and the evil sorcerer has found out that it was you who set the magical creatures free! To confront the sorcerer and protect Pythonia, you need to combine your spells to cast an attack spell to reduce the sorcerer's hit points (HP).

Write a higher-order function called `cast_attack_spell(sum_powers_limit, defeat_threshold)` that takes two parameters that are non-negative numbers and returns **two functions:**

The first function is called `combine_spells(*spells)` which takes an unknown number of spells, where each spell is a tuple of the form `(name, power, effect)` where:

- `name` is a string,
- `power` is a non-negative integer,
- `effect` is a string.

It returns a combined spell tuple of the same format. It follows the given rules for combining spells:

- Rule 1: Spell Name Combination
  - A concatenation of the names of the input spells, separated by **hyphens**, and **sorted** alphabetically, respecting the original letter casing.
  - **Hint 1:** The key parameter in the sorted(), and str.lower() might be helpful, as well as `min` and `sum`
- Rule 2: Power Calculation
  - Sum of the powers of the input spells, but it should not exceed a `sum_powers_limit`. If the sum of powers exceeds it, reduce it to `sum_powers_limit`.
- Rule 3: Effect Determination
  - Determine based on the majority effect among the input spells. If **more than half** of the input spells have the same effect, the combined spell takes on that effect. Otherwise, the effect is **"neutral"**
  - Case sensitivity should be considered when handling effects, i.e. 'fire' and 'Fire' are not the same.

Hint 2:

1. Steps you can take for find a majority:
   a. Create a dictionary, where a key is a given effect (i.e. 'fire', 'ice' etc) and a value of number of occurrences.
   b. Then you can use function max, it takes an optional parameter that specifies what kind of maximum you want to find.

The second function is called `cast_spell(combined_spell, sorcerer_hp)` that takes two parameters:

- `combined_spell`: a tuple, returned by the `combine_spells` function.
- `sorcerer_hp (an integer)`: the current HP of the sorcerer

It reduces the sorcerer's HP by the power of the spell.

- If, after casting the spell, the sorcerer's HP **falls below** `defeat_threshold`, the function should return a string **"Sorcerer is defeated by** `{combined_spell_name}` **with the** `{majority_spell_effect}` **effect."**
  - Where `combined_spell_name` and `majority_spell_effect` are taken from the `combined_spell` parameter.
- If the sorcerer's HP remains at or above `defeat_threshold`, the function should return **"Try again"**

Input	Output	Reason
combine_spells, cast_spell = cast_attack_spell(200, 20) combined = combine_spells(('Fireball', 50, 'fire'), ('Fireball', 60, 'fire')) cast_spell(combined, 40)	Sorcerer is defeated by Fireball-Fireball with the fire effect.	The combined spell name will be `'Fireball-Fireball'` The combined power is 50 + 60 = 110 since it is less than 200 The combined effect will be `'fire'` due to the majority. Therefore, the returned tuple is `('Fireball-Fireball', 110, 'fire)` After cast_spell(combined, 40) is called: `sorcerer_hp` is 40 in this case, so the new hp becomes 40 - 110 = -70 Since -70 < 20, we return the string from the output column.

Requirement:

No requirements here but our solution does not use explicit loops. Try to use everything you have learned to avoid explicit loops as well.

```
def cast_attack_spell(sum_powers_limit, defeat_threshold):
    """
    >>> combine_spells, cast_spell = cast_attack_spell(200, 20)
    >>> combined = combine_spells(('Fireball', 50, 'fire'),
('Fireball', 60, 'fire'))
    >>> print(combined)
    ('Fireball-Fireball', 110, 'fire')
    >>> cast_spell(combined, 40)
    'Sorcerer is defeated by Fireball-Fireball with the fire effect.'

    >>> combined = combine_spells(('Fireball', 50, 'fire'), \
        ('Iceball', 40, 'ice'), ('Lightning', 120, 'lightning'))
    >>> print(combined)
    ('Fireball-Iceball-Lightning', 200, 'neutral')
```

```
    >>> cast_spell(combined, 500)
    'Try again'
    """
```

Good Job! You defeated the evil sorcerer! You saved Pythonia and finished the homework!

Submission

Please submit the homework via Gradescope. You may submit more than once before the deadline, and only the final submission will be graded. Please refer to the Gradescope FAQ if you have any issues with your submission.