LAB 1 (10 points) **Due date:** September 1, 2023 11:59 PM

Before you start…

You will constantly hear me say "Coding is like a sport", how do you get good at something new for you? You practice it until you improve (and then you practice some more). Clearly, there are many ways to approach the practice of coding, and some methods will work better for you than others. But no matter what or how you learn, keep in mind these tips from your instructor:

- Do not compare yourself to others: An assignment might take you hours, compared to other fellow students, but every student learns in a different way.
- Outline a strategy before you start typing code: Going from instructions to code is a common practice among students, but when you are a beginner, this could lead to hours of frustration and anxiety that often result in academic dishonesty. Problem-solving skills are crucial for coding, if you are unable to implement a function, assume the computer understands English and prepare a course of action based on that fact (like pseudocode code or bullet points) so you can discuss it with the course staff or your peers. That is how you learn.
- Be patient: Only you are your worse enemy! When you are stuck ask us for help. Go easy on yourself, set some goals for the day, stick to them, and take breaks, spending a lot of time staring at your screen is not going to solve all your issues.
- Don't forget debugging: Computers are very picky and will only do exactly what you tell them. Tiny typos like incorrect operator or syntax will break your entire program, and it is on you to find all those bugs in your code.
- Write clean code: "It works" is not enough when it comes to coding. In practice, programmers work in teams, so others will be reading your code often. Always do your best to write your code in the most concise and readable way that you can. Start the habit of clean coding now, as it will save you hours trying to decipher your work after you haven't looked at it in months!
- Reminders about dictionaries:
    - Define dictionaries using curly brackets

      ```
      >>> colors = { "Fred": "Blue", "Mary": "Green", "Bob": "Red", "Sara": "Yellow"}
      ```
    - Use the [] operator to access the value associated with a key.

      ```
      >>> colors["Sara"]
       "Yellow"
      ```
    - The in operator is used to test whether a key is in a dictionary.

      ```
      >>> "Daniel" in colors
       False
      ```
    - New items can be added or modified using the [] operator.

      ```
      >>> colors["Daniel"] = "Purple"
      ```

REMINDER: As you work on your coding assignments, it is important to remember that passing the examples provided does not guarantee full credit. While these examples can serve as a helpful starting point, it is ultimately your responsibility to thoroughly test your code and ensure that it is functioning correctly and meets all the requirements and specifications outlined in the assignment instructions. Failure to thoroughly test your code can result in incomplete or incorrect outputs, which will lead to deduction in points for each failed case.

Create additional test cases and share them with you classmates on Teams, we are in this together!

90% > Passing all test cases 10% > Clarity and design of your code

frequency(txt) **3 pts**

Returns a dictionary with the frequency count of each alphabet letter (in lowercase) in txt. You cannot make assumptions about the contents in txt.

```
Preconditions and Postconditions
txt: non-empty str
Returns: dict -> frequency count
```

Useful methods:

- The str.isalpha() method returns True if all characters in the string are alphabetic and there is at least one character, False otherwise.
  - `'s'.isalpha() returns True`
  - `'9'.isalpha() returns False`
- The str.lower() method returns a copy of the string with all the cased characters converted to lowercase.
  - `'A'.lower() returns 'a'`
- ord() returns an integer representing the Unicode code point of that character.
  - `ord('a') returns 97`

You are NOT allowed to use the count() method or any other Python count libraries such as Counter, mode, the min() and max() methods. You will not get credit if you used them in your code, even if your code passed the test cases.

Example:

```
>>> frequency('mama')
{'m': 2, 'a': 2}
>>> answer = frequency('We ARE Penn State!!!')
>>> answer
{'w': 1, 'e': 4, 'a': 2, 'r': 1, 'p': 1, 'n': 2, 's': 1, 't': 2}
# Don't forget dictionaries are unsorted collections
```

invert(d) **4 pts**

Given a dictionary *d*, create a new dictionary that is the invert of *d* such that original key:value pairs *i*:*j* are now related *j*:*i*, but when there are nonunique values (j's) in *d*, they should not be included as a key:value pair in the inverted dictionary. Keys are case sensitive. It returns the new inverted dictionary.

```
Preconditions and Postconditions
d: dict
Returns: dict -> Inverted dictionary mapping value to key, excluding non-unique values
```

** You are NOT allowed to use the zip() or enumerate() methods. You will not get credit if you used them in your code, even if your code passed the test cases.

Examples:

```
>>> invert({'one':1, 'two':2, 'uno':1, 'dos':2, 'three':3})
{3: 'three'}
>>> invert({'one':1, 'two':2,  'three':3, 'four':4})
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
>>> invert({'123-456-78':'Sara', '987-12-585':'Alex', '258715':'sara', '00000':'Alex'})
{'Sara': '123-456-78', 'sara': '258715'}
# Don’t forget dictionaries are unsorted collections
```

Useful methods and operatios:

- List concatenation (+) or append method

employee_update(d, bonus, year) **3 pts**

Modifies the given dictionary d by adding another key:value assignment for all employees but with a bonus for the next year. You can assume the previous year exists in the dictionary.

```
Preconditions and Postconditions
d: dict
bonus: int/float
year: int
Returns: dict -> adds the key:value pair with bonus applied
```

Recommended methods:

- dict.keys(), returns all the keys in a dictionary
  - `D = {'one':1, 'two':2,  'three':3, 'four':4})`
  - `D.keys() returns ['one', 'two', 'three', 'four']`
- List concatenation (+) or append method

Tip: Dictionaries, just like lists, are mutable, so saving a reference to a dictionary, say `x=d[2020]`, mapping `x` to another key in `d` and then changing information in `x`, will also change the data in `d[2020]`, which is not a desirable behavior. When adding a new key, create new dictionaries and lists and populate them with the required data.

Examples:

```
>>> records = {
2020:{"John":["Managing Director","Full-time",65000],"Sally":["HR Director","Full-
time",60000],"Max":["Sales Associate","Part-time",20000]},
2021:{"John":["Managing Director","Full-time",70000],"Sally":["HR Director","Full-
time",65000],"Max":["Sales Associate","Part-time",25000]}}

>>> employee_update(records,7500,2022)
{2020: {'John': ['Managing Director', 'Full-time', 65000], 'Sally': ['HR Director', 'Full-
time', 60000], 'Max': ['Sales Associate', 'Part-time', 20000]},
2021: {'John': ['Managing Director', 'Full-time', 70000], 'Sally': ['HR Director', 'Full-
time', 65000], 'Max': ['Sales Associate', 'Part-time', 25000]},
2022: {'John': ['Managing Director', 'Full-time', 77500], 'Sally': ['HR Director', 'Full-
time', 72500], 'Max': ['Sales Associate', 'Part-time', 32500]}}
```