 **Python Programming**

Python Classes and Data Structures: Practice Problems + Reading

In this homework, you're going to write code for two challenge problems.

You'll practice these programming concepts we've covered in class:

- Using dictionaries to solve problems.
- Storing data in tuples and sets.

Deliverables

Part of this homework will be code challenges and part of it will be some reading with comprehension questions.

For each of the code challenges listed below, you will create a new `.py` file and write code to solve the problem. For example, you would create `problem1.py` with your solution code to the first problem. Run the file from the command line to check your work.

Reminder: On your laptop, you can run the file from your command line with the following:

```
python problem1.py
```

> **Hint:** Make sure you are printing something out with the `print` statement. Otherwise, you won't see any output from running your program!

Requirements:

By the end of this, you should have: * Two different `.py` files (one for each code challenge).

Code Challenges

Problem 1: I Love You, Tuple

Skill you're practicing: Using tuples.

You may recall tuples from your in-class lesson. Tuples are *immutable* data structures, which means they can't be changed after they're created. They are typically used to store related information that doesn't need to be changed, such as a student record or, in this case, some stats about a movie. It's your job to create some `print` statements that access the appropriate values inside each tuple to produce the expected output.

Starter Code

```
# My favorite romance movies
# title, release year, runtime, tagline, main characters
romantic_movie1 = ("The Princess Bride", 1987, 98, "The story of a man and a woman
who lived happily ever after.", ["Buttercup", "Westley", "Fezzik", "Inigo Montoya",
"Vizzini"])
romantic_movie2 = ("Groundhog Day", 1993, 101, "He's having the day of his life…
over and over again.", ["Phil Connors"])
romantic_movie3 = ("Amélie", 2001, 122, "One person can change your life forever.",
["Amélie Poulain", "Nino Quincampoix", "The Garden Gnome"])
```

Expected Output

```
Here are my favorite romance movies:

The Princess Bride (1987): The story of a man and a woman who lived happily ever
after.

Groundhog Day (1993): He's having the day of his life...over and over again.

Amélie (2001): One person can change your life forever.
```

Hint: Remember, you can access a tuple sort of like a list, with the square brackets `[]` counting from zero (e.g., the value of `romantic_movie1[0]` is `"The Princess Bride"`).

Bonus:

Change the `print` statement's separator character to an empty string `""` instead of a space so that you can print the year as `(1987)` instead of `( 1987 )`. Use your toolkit!

Problem 2: Friends, Colleagues, and Details

Skill you're practicing: Using dictionaries, lists, and key-value pairs.

Your boss tasks you with creating a company directory. Make a list called `employees`, which will

contain one dictionary per person and include the keys `name`, `age`, `department`, `phone`, and `salary`. Once you have the list of dictionaries set up, loop through the list and print out the `name`, `department`, and `phone` number of each employee. Their `age` and `salary` should remain secret!

Starter Code

A dictionary should be set up in the following way:

```
{
    "name": "Ron Swanson",
    "age": 55,
    "department": "Management",
    "phone": "555-1234",
    "salary": "$87,000"
}
```

Expected Output

```
Ron Swanson in Management can be reached at 555-1234.
Leslie Knope in Middle Management can be reached at 555-4321.
Andy Dwyer in Shoe Shining can be reached at 555-1122.
April Ludgate in Administration can be reached at 555-3345.
```

Hint: Dictionaries have values that can be accessed with keys, for example, `employees["name"]`. Keys are typically strings.

Reverse Lookup

Overview:

Finding the value from a key is easy: `my_dictionary[key]`. But what if you only have the value and want to find the key?

There's no built-in function for this - you'll be writing it here.

You will practice these programming concepts we've covered in class: * Functions * Loops * Dictionaries

Deliverables

One `.py` file with code that solves the problem.

Requirements

You task is to write a function that takes a dictionary and a value, returning the corresponding key.

For example:

```
state_capitals = {
  "Alaska" : "Juneau",
  "Colorado" : "Denver",
  "Oregon" : "Salem",
  "Texas" : "Austin"
  }

print(reverse_lookup("Denver"))
# Prints Colorado
```

 **Python Part Time**

Equal Sets

Overview:

For two sets to be equal, they simply have to contain the same elements - it doesn't matter what order they're in.

Unfortunately in Python, comparing two sets using == will only produce `True` if the elements are in the same order - not what we want!

There's no built-in function for this - so you'll be writing it here.

You will practice these programming concepts we've covered in class: - Functions - Sets

Deliverables

One `.py` file with code that solves the problem.

Requirements:

Write a function that takes two sets and returns `True` if they have the same elements, even if they aren't in the same order.

Here is an example - try running this normally:

```
fruits = ['orange', 'pear', 'kiwi', 'apple', 'banana']
```

```
fruits_copy = ['orange', 'pear', 'kiwi', 'apple', 'banana']

fruits_reordered = ['pear', 'apple', 'kiwi', 'orange', 'banana']

print("Copy comparison", fruits == fruits_copy)
print("Reordered comparison", fruits == fruits_reordered)
```

By default, the second comparison fails. Your function would instead return `True`.