**Homework 1, Part 1: Python Concepts**

1. **Explain what it means for Python to be both strongly and dynamically typed, and give a concrete example of each**

Strongly typed means that you can't perform operations that are invalid for a data type. For example, you can't add a string and an integer in Python. You would get an error message saying that you can't perform that operation on those data types.

Dynamically typed means that you can perform operations on data types that are not specified when you write your code. For example, you could write a function that takes an input and adds 1 to it, and it would work for any data type that can be added, like integers, floats, and strings.

1. **You would like to create a dictionary that maps some 2-dimensional points to their associated names. As a first attempt, you write the following code:  
   points\_to\_names = {[0, 0]: "home", [1, 2]: "school", [-1, 1]: "market"}  
   However, this results in a type error. Describe what the problem is, and propose a solution**.

A dictionary in Python must have keys that are immutable types. A list is a mutable type, so it cannot be used as a key in a dictionary.

The solution would be using tuple instead: points\_to\_names = {(0, 0): "home", (1, 2): "school", (-1, 1): "market"}

1. **Consider the following two functions, each of which concatenates a list of strings:  
   def concatenate1(strings):  
   *result = ""  
   for s in string:  
   result += s  
   return result  
   def concatenate2(strings):  
   return "".join(strings)*  
   One of these approaches is significantly faster than the other for large inputs. Which version is better, and what is the reason for the discrepancy?**

The second version is significantly faster because it uses the built-in join method, which is optimized for concatenating strings. The first version has to create a new string and concatenate each element to it, which is less efficient.