DATA TYPES AND STRUCTURES — ASSIGNMENT SOLUTIONS

THEORY QUESTIONS

1. What are data structures, and why are they important?
Data structures are organized ways of storing and managing data (e.g., lists, sets, dictionaries, trees).
They are important because they allow efficient data storage, retrieval, and manipulation depending on the problem.

2. Difference between mutable and immutable data types with examples.
- Mutable: Can be changed after creation (list, dict, set).
  Example: l = [1,2]; l[0] = 5 → [5,2]
- Immutable: Cannot be changed after creation (int, str, tuple).
  Example: s = "hi"; s[0] = 'H' → Error.

3. Main differences between lists and tuples in Python.
- List: Mutable, dynamic size, [] syntax.
- Tuple: Immutable, fixed size, () syntax.

4. How dictionaries store data?
Dictionaries store key-value pairs using a hash table. Keys are hashed to compute storage location.

5. Why use a set instead of a list?
Sets automatically remove duplicates and provide faster membership tests (O(1) average).

6. What is a string in Python, and how is it different from a list?
A string is an immutable sequence of characters. A list can store any data type and is mutabl

7. How do tuples ensure data integrity?
Tuples are immutable, meaning values cannot be changed once assigned — ensuring fixed data.

8. What is a hash table, and how does it relate to dictionaries in Python?
A hash table is a data structure mapping keys to values using a hash function. Python dictionaries are implemented using hash tables.

9. Can lists contain different data types in Python?
Yes, lists can contain mixed types: [1, "hello", 3.14].

10. Why are strings immutable in Python?
Because they are stored in memory as fixed sequences for efficiency and security. Any change creates a new string.

11. Advantages of dictionaries over lists?
- Fast lookup by key (O(1) average).
- Store data in key-value pairs for clear relationships.

12. Scenario for tuple over list?
Coordinates (x,y) or fixed configuration values, where modification should not be allowed.

13. How do sets handle duplicate values?
They ignore duplicates automatically: {1,2,2,3} = {1,2,3}.

14. How does the “in” keyword work differently for lists and dictionaries?
- List: checks values in sequence.
- Dictionary: checks for keys.

15. Can you modify the elements of a tuple?
No, tuples are immutable. To "modify", a new tuple must be created.

16. What is a nested dictionary? Example.
A dictionary containing another dictionary.
Example: student = {"name":"Alice","marks":{"math":90,"eng":85}}

17. Time complexity of accessing elements in a dictionary?
Average O(1) due to hashing.

18. When are lists preferred over dictionaries?
When order matters or when indexing by position is needed.

19. Why are dictionaries unordered, and how does it affect retrieval?
Traditionally unordered because hashing does not guarantee sequence. From Python 3.7+, dictionaries preserve insertion order, but retrieval is still by key not index.

20. Difference between a list and a dictionary in data retrieval?
- List: retrieved by index position.
- Dictionary: retrieved by key.

PRACTICAL QUESTIONS – Python Solutions
---

1. Create a string with your name and print it:
```
name = "Alice"
print(name)
```

2. Length of "Hello World":
```
print(len("Hello World"))
```

3. Slice first 3 characters from "Python Programming":
```
s = "Python Programming"
print(s[:3])
```

4. Convert "hello" to uppercase:
```
print("hello".upper())
```

5. Replace "apple" with "orange":
```
s = "I like apple"
print(s.replace("apple","orange"))
```

6. Create a list with numbers 1 to 5:
```
l = [1,2,3,4,5]
print(l)
```

7. Append 10 to list [1,2,3,4]:
```
l = [1,2,3,4]
l.append(10)
print(l)
```

8. Remove 3 from list [1,2,3,4,5]:
```
l = [1,2,3,4,5]
l.remove(3)
print(l)
```

9. Access second element in ['a','b','c','d']:
```
l = ['a','b','c','d']
print(l[1])
```

10. Reverse [10,20,30,40,50]:
```
l = [10,20,30,40,50]
l.reverse()
print(l)
```

11. Create a tuple (100,200,300):
```
t = (100,200,300)
print(t)
```

12. Access second-to-last element:
```
t = ('red','green','blue','yellow')
print(t[-2])
```

13. Find minimum in (10,20,5,15):
```
t = (10,20,5,15)
print(min(t))
```

14. Find index of "cat":
```
t = ('dog','cat','rabbit')
print(t.index("cat"))
```

15. Tuple with fruits, check "kiwi":
```
t = ('apple','banana','cherry')
print("kiwi" in t)
```

16. Create set {'a','b','c'}:
```
s = {'a','b','c'}
print(s)
```

17. Clear set {1,2,3,4,5}:
```
s = {1,2,3,4,5}
s.clear()
print(s)
```

18. Remove 4 from set {1,2,3,4}:
```
s = {1,2,3,4}
s.remove(4)
print(s)
```

19. Union of sets {1,2,3} and {3,4,5}:
```
print({1,2,3} | {3,4,5})
```

20. Intersection of sets {1,2,3} and {2,3,4}:
```
print({1,2,3} & {2,3,4})
```

21. Create dictionary with keys:
```
d = {"name":"Alice","age":20,"city":"Paris"}
print(d)
```

22. Add new key "country":
```
d = {'name':'John','age':25}
d['country'] = 'USA'
print(d)
```

23. Access "name" in dict:
```
d = {'name':'Alice','age':30}
print(d['name'])
```

24. Remove key "age":
```
d = {'name':'Bob','age':22,'city':'New York'}
del d['age']
print(d)
```

25. Check if "city" exists:
```
d = {'name':'Alice','city':'Paris'}
print('city' in d)
```

26. Create list, tuple, dictionary:
```
l = [1,2,3]
```

```
t = (1,2,3)
d = {"a":1,"b":2}
print(l,t,d)
```

27. List of 5 random numbers sorted:
```
import random
l = random.sample(range(1,101),5)
l.sort()
print(l)
```

28. List of strings, print 3rd index:
```
l = ["a","b","c","d","e"]
print(l[3])
```

29. Combine two dictionaries:
```
d1 = {"a":1,"b":2}
d2 = {"c":3,"d":4}
d1.update(d2)
print(d1)
```

30. Convert list of strings to set:
```
l = ["a","b","c","a"]
print(set(l))
```