Python Programming Assignment -4

PAGE No. / DATE / /

1) What is the difference between List and tuple in python ?
Explain in terms of:
- mutability
- memory
- performance
- use cases

→

List - List is the ordered collection of element which is enclosed in square bracket and comma separated value is called list.

Tuple - Tuple is similar to a list but is used when the data should not be changed after creation.

i) Mutability -

List - i) list are mutable
ii) you can add, remove or modify element after creation.

```
lst = [1,2,3]
lst[0] = 10
Print(lst)    # [10,2,3]
```

Tuple - i) Tuples are immutable
ii) once, created elements cannot be changed.

```
tup = (1,2,3)
tup[0] = 10    # TypeError
```

PAGE No.
DATE / /

ii) Memory-

List- i) List requires more memory
ii) stores extra space to allow dynamic resizing

tuple- i) requires less memory
ii) fixed size, no extra memory for modification.

Ex.

```
import sys
Print(sys.getsizeof([1,2,3])    # 88
Print(sys.getsizeof((1,2,3))    # 64
```

iii) Performance-

List- i) slightly slower
ii) Because python has to manage dynamic resizing and mutability.

Tuple- i) faster than list
ii) Because of immutability and simpler internal structure.

iv) use cases-

List- i) when data needs to change frequently.
ii) Ex.
- storing user inputs
- shopping cart items
- Dynamic allocation

PAGE No.
DATE / /

Tuple - i) when data should remain constant
ii) Ex.
- coordinates (x, y)
- fixed configuration values
- dictionary keys

2) Why tuples are faster than lists ?
In which real world scenario would you prefer a tuple a tuple over a list ?

→

Why are tuples faster than lists ?

Tuples may faster than list mainly because of their immutability and simpler internal structure.

i) immutability -
- Tuples are immutable, meaning their values cannot change after creation.
- python code does not need to allocate ~~extra~~ memory for resizing or modifications.

ii) simple memory structure -
- Tuples have a fixed size
- lists store extra information to allow dynamic growth.

iii) faster iterations and Access -
- Because tuples are fixed and simpler, iteration and indexing slightly faster than lists.
- python can ~~internal optimization~~ apply internal optimization to tuples.

PAGE No.
DATE / /

Real world scenario where tuples are prefered over list-

i) Fixed configuration data-
- when values must not change

```
DB_config = ("localhost", 3306, "admin")
```

ii) coordinates / Geographical data-

- coordinates are fixed values.

```
location = (19.0760, 72.8777) # longitude, lattitude
```

iv) Dictionary keys-

- only immutabe objects are dictionary keys
- tuples are commonly used

```
Points = {(1,2): "A", (3,4): "B"}
```

iv) function return values -
- when returning multiple values that should not be changed.

Ex. def get_user():
    return ("sudarshan", 21, "India")

PAGE No.
DATE / /

3) Predict the output:

```
lst = [10, 20, 30]
tup = (10, 20, 30)

lst[0] = 100
tup[0] = 100
```

which line raise an error and why?

→

```
lst = [10, 20, 30]
tup = (10, 20, 30)

lst[0] = 100          #
print(lst)            # [100, 20, 30]

tup[0] = 100          # ERROR.
```

The line tup[0] = 100 giver an error because tuples are immutable in nature
- once created then cannot be changed or modified later.

4) Explain why strings are immutable in python? What happens internally when you modify a string variable?

→

Strings in python are immutable, meaning their state cannot be modified after they are created. This design choise provides several benifits, including improved performance and thread safty, and allowing strings to be used as

dictionary keys. when you modify a string variable, python actually creates a completely new string object in memory.

```
s1 = "Hello"
print(id(s1))        # 1400407765 10128

s1 = s1 + "world"
print(id(s1))        # 1400040770856752
```

It gets different memory addresses.

i) s1 initially points to a string object containing "Hello".

ii) The operation s1 + "world" creates a new string object with the value "Hello world".

iii) The variable name s1 is then updated to point this new object's memory location. The original "Hello" object remains in memory until garbage collected.

5) predict the output:

```
s = "python"
print(id(s))
s = s + "3"
print(id(s))
```

Explain the reason for change no change in id().

→

PAGE No.
DATE / /

→ s = "python"
print(id(s)) # 1653656578224

s = s + "3"
print(id(s)) # 1653658859536

It gets the two different memory Addresses.

Reason for change-

- Strings are immutable in python.
- When we create a string in python it cannot be changed or modified later.
- If we try to update like above s = s + "3" then python create completely new string and points to the different memory location.

6) What is a dictionary in python?
Explain:
- key-value pair
- why keys must be immutable
- why duplicate keys are not allowed.

→ A dictionary in python is a built-in, unordered collection of data that stores items in the form of key value pairs. It is a mutable data structure.
- meaning its contents can be changed after creation and it is designed for efficient retrival and modification of data.
- written using curly braces {}
- Each element is written as key : value

PAGE No.
DATE / /

Ex.
student = {"nome": "sudonshan", "age": 21}

i) key-value pairs-

what is key-
- A key acts like unique identifier
- use to access the value
- must be unique

what is value-
- the data associated with key
- can be of any data type.

ii) why keys must be immutable-
- dictionarey use a hash table internally.
- when key is inserted:
  - python computes the hash value
  - stores the value based on that hash

X what if keys where mutable?
- If a key change after insertion:
  - Hash value would change
  - python would lose track of where the value is stored.

iii) why are duplicate keys not allowed?

- Each key maps exactly one value
- duplicate keys would create ambiguity

Ex.

```
data = { "a" : 1, "a" : 2}
print (data)
```

o/p → {"a" : 2}.

7) Predict the output:

```
d = { 1 : "one", 1 : "ONE", 2 : "TWO" }
print(d)
```

Why does this happen?

→

```
d = { 1 : "one", 1 : "ONE", 2 : "TWO"}
print(d)
```

o/p → { 1 : "ONE", 2 : "TWO"}.

Why this happen?

i) Dictionary keys must be unique.
- In python, a dictionary cannot have duplicate keys
- the key 1 appears twice in the dictionary.

ii) Latest values overwrites the previous value
- when python encounters the second 1 : "ONE"
  - The key 1 already exists
  - Python replaces the old value "one" with "ONE".

PAGE No.
DATE / /

iii) Internal reason (hashing) -
- Dictionaries use hashing.
- Both keys 1 have the same has value.
- so python keeps only one entry per user.

8) What is the range data type in python? How it is different from a list of numbers?

→

range -
The range data type represents a sequence of numbers and is commonly used in loops for iterations.

- represent sequence of numbers
- commonly used in loops
- Generates values on demand.

```
r = range(1,5)
print(list(r))      # [1,2,3,4]
print(type(r))      # <class 'range'>
```

How it is different from a list of numbers -

- A range does not store all values (numbers) in memory.
- It only remembers the start, stop and step values.
- Numbers are generated only when needed.

PAGE No.
DATE / /

A list on other hand:
- stores all numbers at once
- Takes more memory
- can be modified.

9) predict the output:

```
r = range(5)
print(r)
print(list(r))
```

→

```
r = range(5)
print(r)          # range(0,5)
print(list(r))    # [0, 1, 2, 3, 4]
```

The output of the above program is -

o/p →

```
range(0,5)
[0, 1, 2, 3, 4].
```

10) What is difference between:

```
range(1, 10)
range(1, 10, 2)
```

Explain the parameters.

→

range (1, 10) -

- start = 1 → sequence starts from 1
- stop = 10 → sequence stops before 10
- step = 1 → default step value.

It generates -

1, 2, 3, 4, 5, 6, 7, 8, 9

range(1, 10, 2) -

- start = 1 → sequence start from 1
- stop = 10 → sequence stops before 10
- step = 2 → number increases by 2 each time

It generates -

1, 3, 5, 7, 9

Explanation -

Start - The integer at which the sequence begen (inclusive). If ommited (when only one argument is provided), it defaults to 0. In the the examples the start is 1.

Stop - The integer before which the sequence must end (exclusive). The sequence stops before reaching this value. In both the examples, the stop is 10.

PAGE No.
DATE / /

step-

The increment used between in the numbers in the sequence (cannot be zero). if ommited, it defaultr to 1.
range(1,10) uses the default step of 1, while range(1,10,2) explicitly sets the step to 2.