TYPE-B:TUPLES:CH-12

1)(a) Find the output generated by following code fragments :

```
plane = ("Passengers", "Luggage")
plane[1] = "Snakes"
```

sol:
it rises an error , as the statement trying to change the value of the particular index but we know that tuples are immutable.

1)(b) Find the output generated by following code fragments :

```
(a, b, c) = (1, 2, 3)
```

sol:
there will be no output as there is no print statement but the assignment of values occurs successfully ,i.e: a=1,b=2,c=3 (tuple unpacking method)

1)(c) Find the output generated by following code fragments :

```
(a, b, c, d) = (1, 2, 3)
```

sol:
it rises an value error. according to unpacking rule there must be equal number of variable or numerals in both the side but here lhs is not equal to rhs and it is not enough for tuple unpacking.

1)(d) Find the output generated by following code fragments :

```
a, b, c, d = (1, 2, 3)
```

sol:
it rises an value error. according to unpacking rule there must be equal number of variable or numerals in both the side but here lhs is not equal to rhs and it is not enough for tuple unpacking.

1)(e) Find the output generated by following code fragments :

```
a, b, c, d, e = (p, q, r, s, t) = t1
```

sol:

```
ERROR!
Traceback (most recent call last):
 File "<main.py>", line 3, in <module>
NameError: name 't1' is not defined
```

1)(f) a, b, c, d, e = (p, q, r, s, t) = t1

What will be the values and types of variables a, b, c, d, e, p, q, r, s, t if t1 contains (1, 2.0, 3, 4.0, 5) ?

sol:
This uses nested tuple unpacking. First, (p, q, r, s, t) = t1 assigns the values of t1 in order: p = 1, q = 2.0, r = 3, s = 4.0, t = 5. Then, a, b, c, d, e = (p, q, r, s, t) assigns the same values to a, b, c, d, e.

So, all variables a, b, c, d, e, p, q, r, s, t hold the corresponding values from t1, and their types (int or float) are preserved.

1)(g)
```
t2 = ('a')
type(t2)
```

sol:
```
<class 'str'>
```

1)(h) Find the output generated by following code fragments :

```
t3 = ('a',)
type(t3)
```

sol:
```
<class 'tuple'>
```

1)(i)
```
T4=(17)
type(T4)
```

sol:
```
<class 'int'>
```

1)(j)
```
T5=(17,)
type(T5)
```

sol:
```
<class 'tuple'>
```

1)(k)
```
tuple = ( 'a' , 'b',  'c' , 'd' , 'e')
tuple = ( 'A', ) + tuple[1: ]
print(tuple)
```

sol:
```
('A', 'b', 'c', 'd', 'e')
```

1)(l)
```
t2 = (4, 5, 6)
t3 = (6, 7)
t4 = t3 + t2
t5 = t2 + t3
print(t4)
print(t5)
```

sol:
```
(6, 7, 4, 5, 6)
(4, 5, 6, 6, 7)
```

1)(m)
```
t3 = (6, 7)
t4 = t3 * 3
t5 = t3 * (3)
print(t4)
print(t5)
```

sol:
```
(6, 7, 6, 7, 6, 7)
(6, 7, 6, 7, 6, 7)
```

1)(n)
```
t1 = (3,4)
t2 = ('3' , '4')
print(t1 + t2 )
```

sol:
```
(3, 4, '3', '4')
```

1)(o)
```
perc = (88,85,80,88,83,86)
a = perc[2:2]
b = perc[2:]
c = perc[:2]
d = perc[:-2]
e = perc[-2:]
f = perc[2:-2]
g = perc[-2:2]
h = perc[:]
```

sol:
no output

2) What does each of the following expressions evaluate to? Suppose that T is the tuple containing :
("These", ["are", "a", "few", "words"], "that", "we", "will", "use")

sol:

(a) T[1] [0::2]
```
['are', 'few']
```
(b) "a" in T[1][0]
```
True
```
(c) T[:1] +T[1]
Cannot add a tuple and a list directly , This will raise a TypeError

(d) T[2::2]
```
('that', 'will')
```
(e) T[2] [2] in T[1]
```
True
```

3)(a) t = ('a', 'b', 'c', 'd', 'e')
print(t[5])

sol:
it rises the index error as the give range of 5 is ount of range. the tuple has maximum index as 4.

3)(b) t = ('a', 'b', 'c', 'd', 'e')
t[0] = 'A'

sol:
this statement given above rise the Type error as we cannot able to change the element of a tuple in place because tuple is immutable.

3)(c) t1 = (3)
t2 = (4, 5, 6)
t3 = t1 + t2
print (t3)

sol:
the above statements rise the type error because the value of t1 is integer and t2 in a tuple, as the '+' operator undergo its operation only when both the operand are in the same data type, but here they given two different data types with the '+' operator.

3)(d) t1 = (3,)
t2 = (4, 5, 6)
t3 = t1 + t2
print (t3)

sol:
(3,4, 5, 6)

3)(e) t2= (4, 5, 6)
t3= (6, 7)
print(t3 - t2)

sol:
the datatypes except numbers, does not undergo subtraction operator rather it only undergo replication and concatesion.

3)(f) t3 = (6, 7)
t4 = t3 * 3
t5= t3 * (3)
t6 = t3 * (3,)
print(t4)
print(t5)
print(t6)

sol:
line 4 causes an error because in replication operator we must use an integer data type

and as a second operand we can use anyother operators such as string list and tuples;

3)(g) odd=1,3,5
print(odd+[2,4,6]) [4]

sol:

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 2, in <module>
TypeError: can only concatenate tuple (not "list") to tuple
```

3)(h) t = ('a', 'b', 'c', 'd', 'e')
1, 2, 3, 4, 5, = t

sol:

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 2
    1, 2, 3, 4, 5, = t
    ^
SyntaxError: cannot assign to literal
```

it arises as the identifiers cannot only be a integer rather it can be with any other variables.

3)(i) t = ('a', 'b', 'c', 'd', 'e')
1n, 2n, 3n, 4n, 5n = t

sol:

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 2
    1n, 2n, 3n, 4n, 5n = t
    ^
SyntaxError: invalid decimal literal
```

it arises as the identifiers cannot start with integers.

3)(j) t = ('a', 'b', 'c', 'd', 'e')
x, y, z, a, b = t

sol:
the code executed successfully, and the values of a,b,c,d,e is assigned to x,y,z,a,b respectively.

3)(k) t = ('a', 'b', 'c', 'd', 'e')
a,b,c,d,e,f=t

sol:
it arises error accoeding to unpacking of tuple the number of elements and identifiers should

be equal .i.e:lhs = rhs. but the above statement it id not equal.

4) What would be the output of following code if

```
 ntpl = ("Hello", "Nita", "How's", "life?")

(a, b, c, d) = ntpl
print ("a is:", a)
print ("b is:", b)
print ("c is:", c)
print ("d is:", d)
ntpl = (a, b, c, d)
print(ntpl[0][0]+ntpl[1][1], ntpl[1])
```

sol:

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 1, in <module>
NameError: name 'ntpl' is not defined
```

5) Predict the output.

```
tuple_a = 'a', 'b'
tuple_b = ('a',  'b')
print (tuple_a == tuple_b)
```

sol:

```
True
```

6) Find the error. Following code intends to create a tuple with three identical strings. But even after successfully executing following code (No error reported by Python), The len( ) returns a value different from 3. Why ?

```
tup1 = ('Mega') * 3
print(len(tup1))
```

sol:

Problem:

('Mega') is not a tuple, it is just a string.
Multiplying it by 3 repeats the string, giving "MegaMegaMega".
So len(tup1) returns 12 (number of characters), not 3.

Correct way to create a tuple with three identical strings:

```
tup1 = ('Mega',) * 3
print(tup1)      # ('Mega', 'Mega', 'Mega')
print(len(tup1))  # 3
```

Explanation:
The comma after "Mega" makes it a tuple with one element.

Multiplying the tuple by 3 repeats the element 3 times.

7) Predict the output.

```
tuple1 = ('Python') * 3
print(type(tuple1))
```

sol:

```
<class 'str'>
```

8) Predict the output.

```
tup1=(1,)*3
tup1[0]=2
print(tup1)
```

sol:

```
ERROR!
Traceback (most recent call last):
  File "<main.py>", line 2, in <module>
TypeError: 'tuple' object does not support item assignment
```

9) What will be the output of the following code snippet?

```
Tup1 = ((1, 2),) * 7
print(len(Tup1[3:8]))
```

sol:

```
4
```