**Question 1**

Why are tuples called immutable types?

***Answer***

Tuples are called immutable types because we cannot change elements of a tuple in place.

**Question 2**

What are mutable counterparts of tuple?

***Answer***

Lists are mutable counterparts of tuple.

**Question 3**

What are different ways of creating a tuple?

***Answer***

Tuples can be created by the following ways:

1. By placing a sequence of values separated by comma within parentheses.  
   Example:  
   tup = (1, 2, 3)
2. By placing a sequence of values separated by comma without parentheses (parentheses are optional).  
   Example:  
   tup = 1, 2, 3
3. By using the built-in tuple type object (tuple( )) to create tuples from sequences as per the syntax given below:  
   T = tuple(<sequence>)  
   where sequence can be any type of sequence object like strings, tuples, lists, etc.  
   Example:  
   tup = tuple([1, 2, 3, 4])

**Question 4**

What values can we have in a tuple? Do they all have to be the same type\*?

***Answer***

No, all the elements of the tuple need not be of the same type. A tuple can contain elements of all data types.

**Question 5**

How are individual elements of tuples accessed?

***Answer***

The individual elements of a tuple are accessed through their indexes given in square brackets as shown in the example below:

Example:

tup = ("python", "tuple", "computer")

print(tup[1])

**Output**

tuple

**Question 6**

How do you create the following tuples?

(a) (4, 5, 6)

(b) (-2, 1, 3)

(c) (-9, -8, -7, -6, -5)

(d) (-9, -10, -11, -12)

(e) (0, 1, 2)

***Answer***

(a) tup = (4, 5, 6)

(b) tup = (-2, 1, 3)

(c) tup = (-9, -8, -7, -6, -5)

(d) tup = (-9, -10, -11, -12)

(e) tup = (0, 1, 2)

**Question 7**

If a = (5, 4, 3, 2, 1, 0) evaluate the following expressions:

(a) a[0]

(b) a[1]

(c) a[a[0]]

(d) a[a[-1]]

(e) a[a[a[a[2]+1]]]

***Answer***

(a) 5

(b) 4

(c) 0

(d) 5

(e) 1

**Explanation:**

1. The first index of tuple a is 5. Therefore, a[0] ⇒ 5
2. The second index of tuple a is 4. Therefore,  
   a[1] ⇒ 4
3. a[0] represents first index of tuple a which is 5.  
   Now the expression a[a[0]] has become a[5] which implies 0.
4. a[-1] represents last index of tuple a which is 0.  
   Now the expression a[a[-1]] has become a[0] which represents first element of a i.e. 5.
5. a[2] represents third index of tuple a which is 3. Addition of 3 and 1 gives 4. Now the expression a[a[a[a[2]+1]]] has become a[a[a[4]]] where a[4] represents fifth index of tuple a i.e., 1.  
   Now the expression a[a[a[4]]] has become a[a[1]] where a[1] represents second index of tuple a i.e., 4.  
   Now the expression a[a[1]] has become a[4] where a[4] represents fifth index of a i.e. 1.

**Question 8**

Can you change an element of a sequence? What if a sequence is a dictionary? What if a sequence is a tuple?

***Answer***

Yes, we can change any element of a sequence in python only if the type of the sequence is mutable.

1. **Dictionary** — We can change the elements of a dictionary as dictionary is a mutable sequence. For example:  
   d = {'k1':1, 'k2':4}  
   d['k1'] = 2  
   Here, we are changing the value of first key-value pair of dictionary d. Now dictionary d will be : {'k1':2, 'k2':4}
2. **Tuple** — We cannot change the elements of a tuple as tuple is an immutable sequence.  
   For example:  
   tup = (1, 2, 3, 4)  
   tup[0] = 5  
   The above expression will give an error.  
   TypeError: 'tuple' object does not support item assignment

**Question 9**

What does a + b amount to if a and b are tuples?

***Answer***

Given a and b are tuples, so in this case the + operator will work as concatenation operator and join both the tuples.

For example:  
a = (1, 2, 3)  
b = (4, 5, 6)  
c = a + b

Here, c is a tuple with elements (1, 2, 3, 4, 5, 6)

**Question 10**

What does a \* b amount to if a and b are tuples?

***Answer***

If a and b are tuples then a \* b will throw an error since a tuple can not be multiplied to another tuple.

TypeError: can't multiply sequence by non-int of type 'tuple'

**Question 11**

What does a + b amount to if a is a tuple and b is 5?

***Answer***

If a is tuple and b is 5 then a + b will raise a TypeError because it is not possible to concatenate a tuple with an integer.

For example:  
a = (1, 2)  
b = 5  
c = a + b

**Output**

TypeError: can only concatenate tuple (not "int") to tuple

**Question 12**

Is a string the same as a tuple of characters?

***Answer***

No, a string and a tuple of characters are not the same even though they share similar properties like immutability, concatenation, replication, etc.

A few differences between them are as follows:

1. in operator works differently in both of them. Below example shows this difference. Example:

my\_string = "Hello"

my\_tuple = ('h','e','l','l','o')

print("el" in my\_string)

print("el" in my\_tuple)

**Output**

True

False

2. Certain functions like split( ), capitalize( ), title( ), strip( ), etc. are present only in string and not available in tuple object.

**Question 13**

Can you have an integer, a string, a tuple of integers and a tuple of strings in a tuple?

***Answer***

Yes, it is possible to have an integer, a string, a tuple of integers and a tuple of strings in a tuple because tuples can store elements of all data types.

For example:  
tup = (1, 'python', (2, 3), ('a', 'b'))

**Multiple Choice Questions**

**Question 1**

Which of the following statements will create a tuple:

1. tp1=("a", "b")
2. tp1[2]=("a", "b")
3. tp1=(3)\*3
4. None of these

***Answer***

tp1=("a", "b")

**Reason** — A tuple is created by placing all the items (elements) inside parentheses () , separated by commas.

**Question 2**

Choose the correct statement(s).

1. Both tuples and lists are immutable.
2. Tuples are immutable while lists are mutable.
3. Both tuples and lists are mutable.
4. Tuples are mutable while lists are immutable.

***Answer***

Tuples are immutable while lists are mutable.

**Reason** — Tuples can not be modified where as List can be modified.

**Question 3**

Choose the correct statement(s).

1. In Python, a tuple can contain only integers as its elements.
2. In Python, a tuple can contain only strings as its elements.
3. In Python, a tuple can contain elements of different types.
4. In Python, a tuple can contain either string or integer but not both at a time

***Answer***

In Python, a tuple can contain elements of different types.

**Reason** — A tuple can have any number of items and they may be of different types (integer, float, list, string, etc).

**Question 4**

Which of the following is/are correctly declared tuple(s) ?

1. a = ("Hina", "Mina", "Tina", "Nina")
2. a = "Hina", "Mina", "Tina", "Nina")
3. a = ["Hina", "Mina", "Tina", "Nina"]
4. a = (["Hina", "Mina", "Tina", "Nina"])

***Answer***

a = ("Hina", "Mina", "Tina", "Nina")

**Reason** — A tuple is created by placing elements inside parentheses () , separated by commas.

**Question 5**

Which of the following will create a single element tuple ?

1. (1,)
2. (1)
3. ( [1] )
4. tuple([1])

***Answer***

(1,) and tuple([1])

**Reason** — (1,) To create a tuple with only one element, comma needs to be added after the element, otherwise Python will not recognize the variable as a tuple and will consider it as an integer.  
tuple([1]) tuple() function takes any iterable as an argument and hence when u pass a list it makes the list elements as its values.  
Here, 1 is enclosed in square brackets which signifies list. When [1] is being passed as an argument in a tuple, it will generate a single element tuple with element is equal "1".

**Question 6**

What will be the output of following Python code?

tp1 = (2,4,3)

tp3 = tp1\*2

print(tp3)

1. (4,8,6)
2. (2,4,3,2,4,3)
3. (2,2,4,4,3,3)
4. Error

***Answer***

(2,4,3,2,4,3)

**Reason** — The "\*" operator repeats a tuple specified number of times and creates a new tuple.

**Question 7**

What will be the output of following Python code?

tp1 = (15,11,17,16,12)

tp1.pop(12)

print(tp1)

1. (15,11,16,12)
2. (15,11,17,16)
3. (15,11,17,16,12)
4. Error

***Answer***

Error

**Reason** — As tuples are immutable so they don't support pop operation.

**Question 8**

Which of the following options will not result in an error when performed on types in Python where tp = (5,2,7,0,3) ?

1. tp[1] = 2
2. tp.append(2)
3. tp1=tp+tp
4. tp.sum()

***Answer***

tp1=tp+tp

**Reason** — The "+" operator concatenates two tuples and creates a new tuple. First option will throw an error since tuples are immutable, item assignment not supported in tuples. Second and Fourth option will also throw an error since tuple object has no attribute 'append' and 'sum'.

**Question 9**

What will be the output of the following Python code ?

tp = ()

tp1 = tp \* 2

print(len(tp1))

1. 0
2. 2
3. 1
4. Error

***Answer***

0

**Reason** — Empty tuples multiplied with any number yield empty tuples only.

**Question 10**

What will be the output of the following Python code?

tp = (5)

tp1 = tp \* 2

print(len(tp1))

1. 0
2. 2
3. 1
4. Error

***Answer***

Error

**Reason** — tp is not a tuple and holds an integer value hence object of type 'int' has no len()

**Question 11**

What will be the output of the following Python code?

tp = (5,)

tp1 = tp \* 2

print(len(tp1))

1. 0
2. 2
3. 1
4. Error

***Answer***

2

**Reason** — The "\*" operator performs repetition in tuples. tp1 = tp \* 2 will result in tp1 as (5, 5) so its length will be 2.

**Question 12**

Given tp = (5,3,1,9,0). Which of the following two statements will give the same output?

(i) print( tp[:-1] )  
(ii) print( tp[0:5] )  
(iii) print( tp[0:4] )  
(iv) print( tp[-4:] )

1. (i), (ii)
2. (ii), (iv)
3. (i), (iv)
4. (i), (iii)

***Answer***

(i), (iii)

**Reason** — Both will yield **(5, 3, 1, 9)**. We can use indexes of tuple elements to create tuple slices as per following format : seq = T[start:stop]

**Question 13**

What is the output of the following code ?

t = (10, 20, 30, 40, 50, 50, 70)

print(t[5:-1])

1. Blank output( )
2. (50,70)
3. (50,50,70)
4. (50,)

***Answer***

(50,)

**Reason** — Length of tuple t is 7. t[5 : -1] represents tuple slice t[5 : (7-1)] = t[5 : 6] i.e., the element at index 5. So output is (50,).

**Question 14**

What is the output of the following code?

t = (10, 20, 30, 40, 50, 60, 70)

print(t[5:-1])

1. Blank output( )
2. (10, 20, 30, 40, 50)
3. (10, 30, 50, 70)
4. (10, 20, 30, 40, 50, 60, 70)

***Answer***

(60,)

**Reason** — Length of tuple t is 7. t[5 : -1] represents tuple slice t[5 : (7-1)] = t[5 : 6] i.e., the element at index 5. So output is (60,).  
**Note:** There is a misprint in the options provided in the book.

**Question 15**

Which of the below given functions cannot be used with nested tuples ?

1. index( )
2. count( )
3. max( )
4. sum( )

***Answer***

sum( )

**Reason** — For sum( ) function to work, the tuple must have numeric elements. Since nested tuples will have at least one element as tuple so the sum( ) function will raise a TypeError and will not work.

**Fill in the Blanks**

**Question 1**

Tuples are ***immutable*** data types of Python.

**Question 2**

A tuple can store values of ***all*** data types.

**Question 3**

The ***+*** operator used with two tuples, gives a concatenated tuple.

**Question 4**

The ***\**** operator used with a tuple and an integer, gives a replicated tuple.

**Question 5**

To create a single element tuple storing element 5, you may write t = ( ***5,*** ).

**Question 6**

The ***in*** operator can be used to check for an element's presence in a tuple.

**Question 7**

The ***len( )*** function returns the number of elements in a tuple.

**Question 8**

The ***index( )*** function returns the index of an element in a tuple.

**Question 9**

The ***sorted( )*** function sorts the elements of a tuple and returns a list.

**Question 10**

The ***sum( )*** function cannot work with nested tuples.

**True/False Questions**

**Question 1**

Tuples in Python are mutable.  
***False***

**Question 2**

The elements in a tuple can be deleted.  
***False***

**Question 3**

A Tuple can store elements of different types.  
***True***

**Question 4**

A tuple cannot store other tuples in it.  
***False***

**Question 5**

A tuple storing other tuples in it is called a nested tuple.  
***True***

**Question 6**

A tuple element inside another element is considered a single element.  
***True***

**Question 7**

A tuple cannot have negative indexing.  
***False***

**Question 8**

With tuple( ), the argument passed must be sequence type.  
***True***

**Question 9**

All tuple functions work identically with nested tuples.  
***False***

**Question 10**

Functions max( ) and min( ) work with all types of nested tuples.  
***False***

**Type A : Short Answer Questions/Conceptual Questions**

**Question 1**

Discuss the utility and significance of tuples, briefly.

**Answer**

Tuples are used to store multiple items in a single variable. It is a collection which is ordered and immutable i.e., the elements of the tuple can't be changed in place. Tuples are useful when values to be stored are constant and need to be accessed quickly.

**Question 2**

If a is (1, 2, 3)

1. what is the difference (if any) between a \* 3 and (a, a, a) ?
2. Is a \* 3 equivalent to a + a + a ?
3. what is the meaning of a[1:1] ?
4. what is the difference between a[1:2] and a[1:1] ?

**Answer**

1. a \* 3 ⇒ (1, 2, 3, 1, 2, 3, 1, 2, 3)  
   (a, a, a) ⇒ ((1, 2, 3), (1, 2, 3), (1, 2, 3))  
   So, a \* 3 repeats the elements of the tuple whereas (a, a, a) creates nested tuple.
2. Yes, both a \* 3 and a + a + a will result in (1, 2, 3, 1, 2, 3, 1, 2, 3).
3. This colon indicates **(:)** simple slicing operator. Tuple slicing is basically used to obtain a range of items.  
   tuple[Start : Stop] ⇒ returns the portion of the tuple from index Start to index Stop (excluding element at stop).  
   a[1:1] ⇒ This will return empty list as a slice from index 1 to index 0 is an invalid range.
4. Both are creating tuple slice with elements falling between indexes start and stop.  
   a[1:2] ⇒ (2,)  
   It will return elements from index 1 to index 2 (excluding element at 2).  
   a[1:1] ⇒ ()  
   a[1:1] specifies an invalid range as start and stop indexes are the same. Hence, it will return an empty list.

**Question 3**

Does the slice operator always produce a new tuple ?

**Answer**

No, the slice operator does not always produce a new tuple. If the slice operator is applied on a tuple and the result is the same tuple, then it will not produce a new tuple, it will return the same tuple as shown in the example below:

a = (1, 2, 3)

print(a[:])

Slicing tuple a using a[:] results in the same tuple. Hence, in this case, slice operator will not create a new tuple. Instead, it will return the original tuple a.

**Question 4**

The syntax for a tuple with a single item is simply the element enclosed in a pair of matching parentheses as shown below :  
t = ("a")  
Is the above statement true? Why? Why not ?

**Answer**

The statement is false. Single item tuple is always represented by adding a comma after the item. If it is not added then python will consider it as a string.  
For example:  
t1 = ("a",)  
print(type(t1)) ⇒ tuple  
t = ("a")  
print(type(t)) ⇒ string

**Question 5**

Are the following two assignments same ? Why / why not ?  
1.

T1 = 3, 4, 5

T2 = ( 3, 4 , 5)

T3 = (3, 4, 5)

T4 = (( 3, 4, 5))

**Answer**

1. T1 and T2 are same. Both are tuples. We can exclude/include the parentheses when creating a tuple with multiple values.
2. T3 and T4 are not same. T3 is a tuple where as T4 is a nested tuple.

**Question 6**

What would following statements print? Given that we have tuple= ('t', 'p', 'l')

1. print("tuple")
2. print(tuple("tuple"))
3. print(tuple)

**Answer**

1. print("tuple") ⇒ tuple  
   It will simply print the item inside the print statement as it is of string type.
2. print(tuple("tuple")) ⇒ it will throw error.  
   TypeError: 'tuple' object is not callable  
   This is because the variable "tuple" is being used to define a tuple, and then is being used again as if it were a function. This causes python to throw the error as now we are using tuple object as a function but it is already defined as a tuple.
3. print(tuple) ⇒ ('t', 'p', 'l')  
   It will return the actual value of tuple.

**Question 7**

How is an empty tuple created ?

**Answer**

There are two ways of creating an empty tuple:

1. By giving no elements in parentheses in assignment statement.  
   Example:  
   emptyTuple = ()
2. By using the tuple function.  
   Example:  
   emptyTuple = tuple()

**Question 8**

How is a tuple containing just one element created ?

**Answer**

There are two ways of creating single element tuple:

1. By enclosing the element in parentheses and adding a comma after it.  
   Example: t = (a,)
2. By using the built-in tuple type object (tuple( )) to create tuples from sequences:  
   Example:  
   t = tuple([1])  
   Here, we pass a single element list to the tuple function and get back a single element tuple.

**Question 9**

How can you add an extra element to a tuple ?

**Answer**

We can use the concatenation operator to add an extra element to a tuple as shown below. As tuples are immutable so they cannot be modified in place.

For example:

t=(1,2,3)

t\_append = t + (4,)

print(t)

print(t\_append)

Output:

(1,2,3)

(1,2,3,4)

**Question 10**

When would you prefer tuples over lists ?

**Answer**

Tuples are preferred over lists in the following cases:

1. When we want to ensure that data is not changed accidentally. Tuples being immutable do not allow any changes in its data.
2. When we want faster access to data that will not change as tuples are faster than lists.
3. When we want to use the data as a key in a dictionary. Tuples can be used as keys in a dictionary, but lists cannot.
4. When we want to use the data as an element of a set. Tuples can be used as elements of a set, but lists cannot.

**Question 11**

What is the difference between (30) and (30,) ?

**Answer**

a = (30) ⇒ It will be treated as an integer expression, hence a stores an integer 30, not a tuple.  
a = (30,) ⇒ It is considered as single element tuple since a comma is added after the element to convert it into a tuple.

**Question 12**

When would sum( ) not work for tuples ?

**Answer**  
Sum would not work for the following cases:

* When tuple does not have numeric value.  
  For example:-  
  tup = ("a", "b")  
  tup\_sum = sum(tup)  
  TypeError: unsupported operand type(s) for +: 'int' and 'str'

here, "a" and "b" are string not integers therefore they can not be added together.

* Nested tuples having tuple as element.  
  For example:-

a = (1,2,(3,4))

print(sum(a))

Output:  
TypeError: unsupported operand type(s) for +: 'int' and 'tuple'

Here, tuple 'a' is a nested tuple and since it consist of another tuple i.e. (3,4) it's elements can not be added to another tuple. Hence it will throw an error.

* Tuple containing elements of different data type.  
  For example:-

a = (1,2.5,(3,4),"hello")

print(sum(a))

Output:  
TypeError: unsupported operand type(s) for +: 'float' and 'tuple'

Tuple a contains elements of integer, float, string and tuple type which can not be added together.

**Question 13**

Do min( ), max( ) always work for tuples ?

**Answer**

No, min( ), max( ) does not always work for tuples. For min( ), max( ) to work, the elements of the tuple should be of the same type.

**Question 14**

Is the working of in operator and tuple.index( ) same ?

**Answer**

Both in operator and tuple.index( ) can be used to search for an element in the tuple but their working it is not exactly the same.  
The "in" operator returns true or false whereas tuple.index() searches for a element for the first occurrence and returns its position. If the element is not found in tuple or the index function is called without passing any element as a parameter then tuple.index( ) raises an error:

For Example:-  
tuple = (1, 3, 5, 7, 9)  
print(3 in tuple) ⇒ True  
print(4 in tuple) ⇒ False  
print(tuple.index(3)) ⇒ 1  
print(tuple.index(2)) ⇒ Error

ValueError: tuple.index(x): x not in tuple

print(tuple.index()) ⇒ Error

TypeError: index expected at least 1 argument, got 0

**Question 15**

How are in operator and index( ) similar or different ?

**Answer**

**Similarity:**

in operator and index( ) both search for a value in tuple.

**Difference:**

in operator returns true if element exists in a tuple otherwise returns false. While index( ) function returns the index of an existing element of the tuple. If the given element does not exist in tuple, then index( ) function raises an error.

**Type B: Application Based Questions**

**Question 1(a)**

Find the output generated by following code fragments :

plane = ("Passengers", "Luggage")

plane[1] = "Snakes"

***Answer***

**Output**

TypeError: 'tuple' object does not support item assignment

**Explanation**

Since tuples are immutable, tuple object does not support item assignment.

**Question 1(b)**

Find the output generated by following code fragments :

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

***Answer***

**Output**

a = 1

b = 2

c = 3

**Explanation**

When we put tuples on both sides of an assignment operator, a tuple unpacking operation takes place. The values on the right are assigned to the variables on the left according to their relative position in each tuple. As you can see in the above example, a will be 1, b will be 2, and c will be 3.

**Question 1(c)**

Find the output generated by following code fragments :

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

***Answer***

**Output**

ValueError: not enough values to unpack (expected 4, got 3)

**Explanation**

Tuple unpacking requires that the list of variables on the left has the same number of elements as the length of the tuple. In this case, the list of variables has one more element than the length of the tuple so this statement results in an error.

**Question 1(d)**

Find the output generated by following code fragments :

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

***Answer***

**Output**

ValueError: not enough values to unpack (expected 4, got 3)

**Explanation**

Tuple unpacking requires that the list of variables on the left has the same number of elements as the length of the tuple. In this case, the list of variables has one more element than the length of the tuple so this statement results in an error.

**Question 1(e)**

Find the output generated by following code fragments :

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

***Answer***

**Output**

Assuming t1 contains (1, 2.0, 3, 4.0, 5), the output will be:

a = 1

p = 1

b = 2.0

q = 2.0

c = 3

r = 3

d = 4.0

s = 4.0

e = 5

t = 5

**Explanation**

The statement unpacks the tuple t1 into the two variable lists given on the left of t1. The list of variables may or may not be enclosed in parenthesis. Both are valid syntax for tuple unpacking. t1 is unpacked into each of the variable lists a, b, c, d, e and p, q, r, s, t. The corresponding variables of the two lists will have the same value that is equal to the corresponding element of the tuple.

**Question 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) ?

***Answer***

| **Variable** | **Value** | **Type** |
| --- | --- | --- |
| a | 1 | int |
| b | 2.0 | float |
| c | 3 | int |
| d | 4.0 | float |
| e | 5 | int |
| p | 1 | int |
| q | 2.0 | float |
| r | 3 | int |
| s | 4.0 | float |
| t | 5 | int |

**Explanation**

The statement unpacks the tuple t1 into the two variable lists given on the left of t1. The list of variables may or may not be enclosed in parenthesis. Both are valid syntax for tuple unpacking. t1 is unpacked into each of the variable lists a, b, c, d, e and p, q, r, s, t. The corresponding variables of the two lists will have the same value that is equal to the corresponding element of the tuple.

**Question 1(g)**

Find the output generated by following code fragments :

t2 = ('a')

type(t2)

***Answer***

**Output**

<class 'str'>

**Explanation**

The type() function is used to get the type of an object. Here, 'a' is enclosed in parenthesis but comma is not added after it, hence it is not a tuple and belong to string class.

**Question 1(h)**

Find the output generated by following code fragments :

t3 = ('a',)

type(t3)

***Answer***

**Output**

<class 'tuple'>

**Explanation**

Since 'a' is enclosed in parenthesis and a comma is added after it, so t3 becomes a single element tuple instead of a string.

**Question 1(i)**

Find the output generated by following code fragments :

T4 = (17)

type(T4)

***Answer***

**Output**

<class 'int'>

**Explanation**

Since no comma is added after the element, so even though it is enclosed in parenthesis still it will be treated as an integer, hence T4 stores an integer not a tuple.

**Question 1(j)**

Find the output generated by following code fragments :

T5 = (17,)

type(T5)

***Answer***

**Output**

<class 'tuple'>

**Explanation**

Since 17 is enclosed in parenthesis and a comma is added after it, so T5 becomes a single element tuple instead of an integer.

**Question 1(k)**

Find the output generated by following code fragments :

tuple = ( 'a' , 'b', 'c' , 'd' , 'e')

tuple = ( 'A', ) + tuple[1: ]

print(tuple)

***Answer***

**Output**

('A', 'b', 'c', 'd', 'e')

**Explanation**

tuple[1:] creates a tuple slice of elements from index 1 (indexes always start from zero) to the last element i.e. ('b', 'c', 'd', 'e').  
+ operator concatenates tuple ( 'A', ) and tuple slice tuple[1: ] to form a new tuple.

**Question 1(l)**

Find the output generated by following code fragments :

t2 = (4, 5, 6)

t3 = (6, 7)

t4 = t3 + t2

t5 = t2 + t3

print(t4)

print(t5)

***Answer***

**Output**

(6, 7, 4, 5, 6)

(4, 5, 6, 6, 7)

**Explanation**

Concatenate operator concatenates the tuples in the same order in which they occur to form new tuple. t2 and t3 are concatenated using + operator to form tuples t4 and t5.

**Question 1(m)**

Find the output generated by following code fragments :

t3 = (6, 7)

t4 = t3 \* 3

t5 = t3 \* (3)

print(t4)

print(t5)

***Answer***

**Output**

(6, 7, 6, 7, 6, 7)

(6, 7, 6, 7, 6, 7)

**Explanation**

The repetition operator \* replicates the tuple specified number of times. The statements t3 \* 3 and t3 \* (3) are equivalent as (3) is an integer not a tuple because of lack of comma inside parenthesis. Both the statements repeat t3 three times to form tuples t4 and t5.

**Question 1(n)**

Find the output generated by following code fragments :

t1 = (3,4)

t2 = ('3' , '4')

print(t1 + t2 )

***Answer***

**Output**

(3, 4, '3', '4')

**Explanation**

Concatenate operator + combines the two tuples to form new tuple.

**Question 1(o)**

What will be stored in variables a, b, c, d, e, f, g, h, after following statements ?

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[:]

***Answer***

The values of variables a, b, c, d, e, f, g, h after the statements will be:

a ⇒ ( )  
b ⇒ (80, 88, 83, 86)  
c ⇒ (88, 85)  
d ⇒ (88, 85, 80, 88)  
e ⇒ (83, 86)  
f ⇒ (80, 88)  
g ⇒ ( )  
h ⇒ (88, 85, 80, 88, 83, 86)

**Explanation**

1. perc[2:2] specifies an invalid range as start and stop indexes are the same. Hence, an empty slice is stored in a.
2. Since stop index is not specified, perc[2:] will return a tuple slice containing elements from index 2 to the last element.
3. Since start index is not specified, perc[:2] will return a tuple slice containing elements from start to the element at index 1.
4. Length of Tuple is 6 and perc[:-2] implies to return a tuple slice containing elements from start till perc[ : (6-2)] = perc[ : 4] i.e., the element at index 3.
5. Length of Tuple is 6 and perc[-2: ] implies to return a tuple slice containing elements from perc[(6-2): ] = perc[4 : ] i.e., from the element at index 4 to the last element.
6. Length of Tuple is 6 and perc[2:-2] implies to return a tuple slice containing elements from index 2 to perc[2:(6-2)] = perc[2 : 4] i.e., to the element at index 3.
7. Length of Tuple is 6 and perc[-2: 2] implies to return a tuple slice containing elements from perc[(6-2) : 2] = perc[4 : 2] i.e., index at 4 to index at 2 but that will yield empty tuple as starting index has to be lower than stopping index which is not true here.
8. It will return all the elements since start and stop index is not specified.

**Question 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")

1. T[1][0: :2]
2. "a" in T[1][0]
3. T[:1] + [1]
4. T[2::2]
5. T[2][2] in T[1]

***Answer***

The given expressions evaluate to the following:

1. ['are', 'few']
2. True
3. TypeError: can only concatenate tuple (not "list") to tuple
4. ('that', 'will')
5. True

**Explanation**

1. T[1] represents first element of tuple i.e., the list ["are" , "a", "few", "words"]. [0 : : 2] creates a list slice starting from element at index zero of the list to the last element including every 2nd element (i.e., skipping one element in between).
2. "in" operator is used to check elements presence in a sequence. T[1] represents the list ["are" , "a", "few", "words"]. T[1][0] represents the string "are". Since "a" is present in "are", it returns true.
3. T[:1] is a tuple where as [1] is a list. They both can not be concatenated with each other.
4. T[2::2] creates a tuple slice starting from element at index two of the tuple to the last element including every 2nd element (i.e., skipping one element in between).
5. T[2] represents the string "that". T[2][2] represents third letter of "that" i.e., "a". T[1] represents the list ["are" , "a", "few", "words"]. Since "a" is present in the list, the in operator returns True.

**Question 3(a)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ('a', 'b', 'c', 'd', 'e')

print(t[5])

***Answer***

**Output**

IndexError: tuple index out of range

**Explanation**

Tuple t has 5 elements starting from index 0 to 4. t[5] will throw an error since index 5 doesn't exist.

**Question 3(b)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ('a', 'b', 'c', 'd', 'e')

t[0] = 'A'

***Answer***

**Output**

TypeError: 'tuple' object does not support item assignment

**Explanation**

Tuple is a collection of ordered and unchangeable items as they are immutable. So once a tuple is created we can neither change nor add new values to it.

**Question 3(c)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t1 = (3)

t2 = (4, 5, 6)

t3 = t1 + t2

print (t3)

***Answer***

**Output**

TypeError: unsupported operand type(s) for +: 'int' and 'tuple'

**Explanation**

t1 holds an integer value not a tuple since comma is not added after the element where as t2 is a tuple. So here, we are trying to use + operator with an int and tuple operand which results in this error.

**Question 3(d)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t1 = (3,)

t2 = (4, 5, 6)

t3 = t1 + t2

print (t3)

***Answer***

**Output**

(3, 4, 5, 6)

**Explanation**

t1 is a single element tuple since comma is added after the element 3, so it can be easily concatenated with other tuple. Hence, the code executes successfully without giving any errors.

**Question 3(e)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t2 = (4, 5, 6)

t3 = (6, 7)

print(t3 - t2)

***Answer***

**Output**

TypeError: unsupported operand type(s) for -: 'tuple' and 'tuple'

**Explanation**

Arithmetic operations are not defined in tuples. Hence we can't remove items in a tuple.

**Question 3(f)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t3 = (6, 7)

t4 = t3 \* 3

t5= t3 \* (3)

t6 = t3 \* (3,)

print(t4)

print(t5)

print(t6)

***Answer***

**Output**

TypeError: can't multiply sequence by non-int of type 'tuple'

**Explanation**

The repetition operator \* replicates the tuple specified number of times. The statements t3 \* 3 and t3 \* (3) are equivalent as (3) is an integer not a tuple because of lack of comma inside parenthesis. Both the statements repeat t3 three times to form tuples t4 and t5.  
In the statement, t6 = t3 \* (3,), (3,) is a single element tuple and we can not multiply two tuples. Hence it will throw an error.

**Question 3(g)**

Carefully read the given code fragments and figure out the errors that the code may produce.

odd= 1,3,5

print(odd + [2, 4, 6])[4]

***Answer***

**Output**

TypeError: can only concatenate tuple (not "list") to tuple

**Explanation**

Here [2,4,6] is a list and odd is a tuple so because of different data types, they can not be concatenated with each other.

**Question 3(h)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ( 'a', 'b', 'c', 'd', 'e')

1, 2, 3, 4, 5, = t

***Answer***

**Output**

SyntaxError: cannot assign to literal

**Explanation**

When unpacking a tuple, the LHS (left hand side) should contain a list of variables. In the statement, 1, 2, 3, 4, 5, = t, LHS is a list of literals not variables. Hence, we get this error.

**Question 3(i)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ( 'a', 'b', 'c', 'd', 'e')

1n, 2n, 3n, 4n, 5n = t

***Answer***

**Output**

SyntaxError: invalid decimal literal

**Explanation**

This error occurs when we declare a variable with a name that starts with a digit. Here, t is a tuple containing 5 values and then we are performing unpacking operation of tuples by assigning tuple values to 1n,2n,3n,4n,5n which is not possible since variable names cannot start with numbers.

**Question 3(j)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ( 'a', 'b', 'c', 'd', 'e')

x, y, z, a, b = t

***Answer***

**Output**

The code executes successfully without giving any errors. After execution of the code, the values of the variables are:

x ⇒ a

y ⇒ b

z ⇒ c

a ⇒ d

b ⇒ e

**Explanation**

Here, Python assigns each of the elements of tuple t to the variables on the left side of assignment operator. This process is called Tuple unpacking.

**Question 3(k)**

Carefully read the given code fragments and figure out the errors that the code may produce.

t = ( 'a', 'b', 'c', 'd', 'e')

a, b, c, d, e, f = t

***Answer***

**Output**

ValueError: not enough values to unpack (expected 6, got 5)

**Explanation**

In tuple unpacking, the number of elements in the left side of assignment must match the number of elements in the tuple.  
Here, tuple t contains 5 elements where as left side contains 6 variables which leads to mismatch while assigning values.

**Question 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])

***Answer***

**Output**

a is: Hello

b is: Nita

c is: How's

d is: life?

Hi Nita

**Explanation**

ntpl is a tuple containing 4 elements. The statement (a, b, c, d) = ntpl unpacks the tuple ntpl into the variables a, b, c, d. After that, the values of the variables are printed.

The statement ntpl = (a, b, c, d) forms a tuple with values of variables a, b, c, d and assigns it to ntpl. As these variables were not modified, so effectively ntpl still contains the same values as in the first statement.

ntpl[0] ⇒ "Hello"  
∴ ntpl[0][0] ⇒ "H"

ntpl[1] ⇒ "Nita"  
∴ ntpl[1][1] ⇒"i"

ntpl[0][0] and ntpl[1][1] concatenates to form "Hi". Thus ntpl[0][0]+ntpl[1][1], ntpl[1] will return "Hi Nita ".

**Question 5**

Predict the output.

tuple\_a = 'a', 'b'

tuple\_b = ('a', 'b')

print (tuple\_a == tuple\_b)

***Answer***

**Output**

True

**Explanation**

Tuples can be declared with or without parentheses (parentheses are optional). Here, tuple\_a is declared without parentheses where as tuple\_b is declared with parentheses but both are identical. As both the tuples contain same values so the equality operator ( == ) returns true.

**Question 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))

***Answer***

**Output**

12

**Explanation**

This is because tup1 is not a tuple but a string. To make tup1 a tuple it should be initialized as following:  
tup1 = ('Mega',) \* 3  
i.e., a comma should be added after the element.  
We are getting 12 as output because the string "Mega" has four characters which when replicated by three times becomes of length 12.

**Question 7**

Predict the output.

tuple1 = ('Python') \* 3

print(type(tuple1))

***Answer***

**Output**

<class 'str'>

**Explanation**

This is because tuple1 is not a tuple but a string. To make tuple1 a tuple it should be initialized as following:  
tuple1 = ('Python',) \* 3  
i.e. a comma should be added after the element.

**Question 8**

Predict the output.

x = (1, (2, (3, (4,))))

print(len(x))

print( x[1][0] )

print( 2 in x )

y = (1, (2, (3,), 4), 5)

print( len(y) )

print( len(y[1]))

print( y[2] + 50 )

z = (2, (1, (2, ), 1), 1)

print( z[z[z[0]]])

***Answer***

**Output**

2

2

False

3

3

55

(1, (2,), 1)

**Explanation**

* print(len(x)) will return 2. x is a nested tuple containing two elements — the number 1 and another nested tuple (2, (3, (4,))).
* print( x[1] [0] ) Here, x[1] implies first element of tuple which is (2,(3,(4,))) and x[1] [0] implies 0th element of x[1] i.e. 2 .
* print( 2 in x ) "in" operator will search for element 2 in tuple x and will return ""False"" since 2 is not an element of parent tuple "x". Parent tuple "x" only has two elements with x[0] = 1 and x[1] = (2, (3, (4,))) where x[1] is itself a nested tuple.
* y = (1, (2, (3,), 4), 5) y is a nested tuple containing three elements — the number 1 , the nested tuple (2, (3,), 4) and the number 5. Therefore, print( len(y) ) will return 3.
* print( len(y[1])) will return "3". As y[1] implies (2, (3,), 4). It has 3 elements — 2 (number), (3,) (tuple) and 4 (number).
* print( y[2] + 50 ) prints "55". y[2] implies second element of tuple y which is "5". Addition of 5 and 50 gives 55.
* print( z[z[z[0]]]) will return (1, (2,), 1).  
  z[0] is equivalent to 2 i.e., first element of tuple z.  
  Now the expression has become z[z[2]] where z[2] implies third element of tuple i.e. 1.  
  Now the expression has become z[1] which implies second element of tuple i.e. (1, (2,), 1).

**Question 9**

What will the following code produce ?

Tup1 = (1,) \* 3

Tup1[0] = 2

print(Tup1)

***Answer***

**Output**

TypeError: 'tuple' object does not support item assignment

**Explanation**

(1,) is a single element tuple. \* operator repeats (1,) three times to form (1, 1, 1) that is stored in Tup1.  
Tup1[0] = 2 will throw an error, since tuples are immutable. They cannot be modified in place.

**Question 10**

What will be the output of the following code snippet?

Tup1 = ((1, 2),) \* 7

print(len(Tup1[3:8]))

***Answer***

**Output**

4

**Explanation**

\* operator repeats ((1, 2),) seven times and the resulting tuple is stored in Tup1. Therefore, Tup1 will contain ((1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2), (1, 2)).

Tup1[3:8] will create a tuple slice of elements from index 3 to index 7 (excluding element at index 8) but Tup1 has total 7 elements, so it will return tuple slice of elements from index 3 to last element i.e ((1, 2), (1, 2), (1, 2), (1, 2)).  
len(Tup1[3:8]) len function is used to return the total number of elements of tuple i.e., 4.

**Type C: Programming Practice/Knowledge based Questions**

**Question 1**

Write a Python program that creates a tuple storing first 9 terms of Fibonacci series.

**Solution**

lst = [0,1]

a = 0

b = 1

c = 0

for i in range(7):

c = a + b

a = b

b = c

lst.append(c)

tup = tuple(lst)

print("9 terms of Fibonacci series are:", tup)

**Output**

9 terms of Fibonacci series are: (0, 1, 1, 2, 3, 5, 8, 13, 21)

**Question 2(a)**

Write a program that receives the index and returns the corresponding value.

**Solution**

tup = eval(input("Enter the elements of tuple:"))

b = int(eval(input("Enter the index value:")))

c = len(tup)

if b < c:

print("value of tuple at index", b ,"is:" ,tup[b])

else:

print("Index is out of range")

**Output**

Enter the elements of tuple: 1,2,3,4,5

Enter the index value: 3

value of tuple at index 3 is: 4

**Question 2(b)**

Write a program that receives a Fibonacci term and returns a number telling which term it is. For instance, if you pass 3, it returns 5, telling it is 5th term; for 8, it returns 7.

**Solution**

term = int(input ("Enter Fibonacci Term: "))

fib = (0,1)

while(fib[len(fib) - 1] < term):

fib\_len = len(fib)

fib = fib + (fib[fib\_len - 2] + fib[fib\_len - 1],)

fib\_len = len(fib)

if term == 0:

print("0 is fibonacci term number 1")

elif term == 1:

print("1 is fibonacci term number 2")

elif fib[fib\_len - 1] == term:

print(term, "is fibonacci term number", fib\_len)

else:

print("The term", term , "does not exist in fibonacci series")

**Output**

Enter Fibonacci Term: 8

8 is fibonacci term number 7

**Question 3**

Write a program to input n numbers from the user. Store these numbers in a tuple. Print the maximum and minimum number from this tuple.

**Solution**

n = eval(input("Enter the numbers: "))

tup = tuple(n)

print("Tuple is:", tup)

print("Highest value in the tuple is:", max(tup))

print("Lowest value in the tuple is:", min(tup))

**Output**

Enter the numbers: 3,1,6,7,5

Tuple is: (3, 1, 6, 7, 5)

Highest value in the tuple is: 7

Lowest value in the tuple is: 1

**Question 4**

Write a program to create a nested tuple to store roll number, name and marks of students.

**Solution**

tup = ()

ans = "y"

while ans == "y" or ans == "Y" :

roll\_num = int(input("Enter roll number of student: "))

name = input("Enter name of student: ")

marks = int(input("Enter marks of student: "))

tup += ((roll\_num, name, marks),)

ans = input("Do you want to enter more marks? (y/n): ")

print(tup)

**Output**

Enter roll number of student: 1

Enter name of student: Shreya Bansal

Enter marks of student: 85

Do you want to enter more marks? (y/n): y

Enter roll number of student: 2

Enter name of student: Nikhil Gupta

Enter marks of student: 78

Do you want to enter more marks? (y/n): y

Enter roll number of student: 3

Enter name of student: Avni Dixit

Enter marks of student: 96

Do you want to enter more marks? (y/n): n

((1, 'Shreya Bansal', 85), (2, 'Nikhil Gupta', 78), (3, 'Avni Dixit', 96))

**Question 5**

Write a program that interactively creates a nested tuple to store the marks in three subjects for five students, i.e., tuple will look somewhat like :  
marks( (45, 45, 40), (35, 40, 38), (36, 30, 38), (25, 27, 20), (10, 15, 20) )

**Solution**

num\_of\_students = 5

tup = ()

for i in range(num\_of\_students):

print("Enter the marks of student", i + 1)

m1 = int(input("Enter marks in first subject: "))

m2 = int(input("Enter marks in second subject: "))

m3 = int(input("Enter marks in third subject: "))

tup = tup + ((m1, m2, m3),)

print()

print("Nested tuple of student data is:", tup)

**Output**

Enter the marks of student 1

Enter marks in first subject: 89

Enter marks in second subject: 78

Enter marks in third subject: 67

Enter the marks of student 2

Enter marks in first subject: 56

Enter marks in second subject: 89

Enter marks in third subject: 55

Enter the marks of student 3

Enter marks in first subject: 88

Enter marks in second subject: 78

Enter marks in third subject: 90

Enter the marks of student 4

Enter marks in first subject: 78

Enter marks in second subject: 67

Enter marks in third subject: 56

Enter the marks of student 5

Enter marks in first subject: 45

Enter marks in second subject: 34

Enter marks in third subject: 23

Nested tuple of student data is: ((89, 78, 67), (56, 89, 55), (88, 78, 90), (78, 67, 56), (45, 34, 23))

**Question 6**

Write a program that interactively creates a nested tuple to store the marks in three subjects for five students and also add a function that computes total marks and average marks obtained by each student.  
Tuple will look somewhat like :  
marks( (45, 45, 40), (35, 40, 38),(36, 30, 38), (25, 27, 20), (10, 15, 20) )

**Solution**

num\_of\_students = 5

tup = ()

def totalAndAvgMarks(x):

total\_marks = sum(x)

avg\_marks = total\_marks / len(x)

return (total\_marks, avg\_marks)

for i in range(num\_of\_students):

print("Enter the marks of student", i + 1)

m1 = int(input("Enter marks in first subject: "))

m2 = int(input("Enter marks in second subject: "))

m3 = int(input("Enter marks in third subject: "))

tup = tup + ((m1, m2, m3),)

print()

print("Nested tuple of student data is:", tup)

for i in range(num\_of\_students):

print("The total marks of student", i + 1,"=", totalAndAvgMarks(tup[i])[0])

print("The average marks of student", i + 1,"=", totalAndAvgMarks(tup[i])[1])

print()

**Output**

Enter the marks of student 1

Enter marks in first subject: 25

Enter marks in second subject: 45

Enter marks in third subject: 45

Enter the marks of student 2

Enter marks in first subject: 90

Enter marks in second subject: 89

Enter marks in third subject: 95

Enter the marks of student 3

Enter marks in first subject: 68

Enter marks in second subject: 70

Enter marks in third subject: 56

Enter the marks of student 4

Enter marks in first subject: 23

Enter marks in second subject: 56

Enter marks in third subject: 45

Enter the marks of student 5

Enter marks in first subject: 100

Enter marks in second subject: 98

Enter marks in third subject: 99

Nested tuple of student data is: ((25, 45, 45), (90, 89, 95), (68, 70, 56), (23, 56, 45), (100, 98, 99))

The total marks of student 1 = 115

The average marks of student 1 = 38.333333333333336

The total marks of student 2 = 274

The average marks of student 2 = 91.33333333333333

The total marks of student 3 = 194

The average marks of student 3 = 64.66666666666667

The total marks of student 4 = 124

The average marks of student 4 = 41.333333333333336

The total marks of student 5 = 297

The average marks of student 5 = 99.0

**Question 7**

Write a program that inputs two tuples and creates a third, that contains all elements of the first followed by all elements of the second.

**Solution**

tup1 = eval(input("Enter the elements of first tuple: "))

tup2 = eval(input("Enter the elements of second tuple: "))

tup3 = tup1 + tup2

print(tup3)

**Output**

Enter the elements of first tuple: 1,3,5,7,9

Enter the elements of second tuple: 2,4,6,8,10

(1, 3, 5, 7, 9, 2, 4, 6, 8, 10)

**Question 8**

Write a program as per following specification :

"'Return the length of the shortest string in the tuple of strings str\_tuple.  
Precondition: the tuple will contain at least one element."'

**Solution**

str\_tuple = ("computer science with python" ,"Hello Python" ,"Hello World" ,"Tuples")

shortest\_str = min(str\_tuple)

shortest\_str\_len = len(shortest\_str)

print("The length of shortest string in the tuple is:", shortest\_str\_len)

**Output**

The length of shortest string in the tuple is: 12

**Question 9(a)**

Create a tuple containing the squares of the integers 1 through 50 using a for loop.

**Solution**

tup = ()

for i in range(1,51):

tup = tup + (i\*\*2,)

print("The square of integers from 1 to 50 is:" ,tup)

**Output**

The square of integers from 1 to 50 is: (1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304, 2401, 2500)

**Question 9(b)**

Create a tuple ('a', 'bb', 'ccc', 'dddd', ... ) that ends with 26 copies of the letter z using a for loop.

**Solution**

tup = ()

for i in range(1, 27):

tup = tup + (chr(i + 96)\* i,)

print(tup)

**Output**

('a', 'bb', 'ccc', 'dddd', 'eeeee', 'ffffff', 'ggggggg', 'hhhhhhhh', 'iiiiiiiii', 'jjjjjjjjjj', 'kkkkkkkkkkk', 'llllllllllll', 'mmmmmmmmmmmmm', 'nnnnnnnnnnnnnn', 'ooooooooooooooo', 'pppppppppppppppp', 'qqqqqqqqqqqqqqqqq', 'rrrrrrrrrrrrrrrrrr', 'sssssssssssssssssss', 'tttttttttttttttttttt', 'uuuuuuuuuuuuuuuuuuuuu', 'vvvvvvvvvvvvvvvvvvvvvv', 'wwwwwwwwwwwwwwwwwwwwwww', 'xxxxxxxxxxxxxxxxxxxxxxxx', 'yyyyyyyyyyyyyyyyyyyyyyyyy', 'zzzzzzzzzzzzzzzzzzzzzzzzzz')

**Question 10**

Given a tuple pairs = ((2, 5), (4, 2), (9, 8), (12, 10)), count the number of pairs (a, b) such that both a and b are even.

**Solution**

tup = ((2,5),(4,2),(9,8),(12,10))

count = 0

tup\_length = len(tup)

for i in range (tup\_length):

if tup [i][0] % 2 == 0 and tup[i][1] % 2 == 0:

count = count + 1

print("The number of pair where both a and b are even:", count)

**Output**

The number of pair where both a and b are even: 2

**Question 11**

Write a program that inputs two tuples seq\_a and seq\_b and prints True if every element in seq\_a is also an element of seq\_b, else prints False.

**Solution**

seq\_a = eval(input("Enter the first tuple: "))

seq\_b = eval(input("Enter the second tuple: "))

for i in seq\_a:

if i not in seq\_b:

print("False")

break

else:

print("True")

**Output**

Enter the first tuple: 1,3,5

Enter the second tuple: 4,5,1,3

True

**Question 12**

**Computing Mean.** Computing the mean of values stored in a tuple is relatively simple. The mean is the sum of the values divided by the number of values in the tuple. That is,

xˉ=∑xN;∑x=the sum of x,N=number of elements*x*ˉ=*N*∑*x*​;∑*x*=the sum of *x*,*N*=number of elements

Write a program that calculates and displays the mean of a tuple with numeric elements.

**Solution**

tup = eval(input ("Enter the numeric tuple: "))

total = sum(tup)

tup\_length = len(tup)

mean = total / tup\_length

print("Mean of tuple:", mean)

**Output**

Enter the numeric tuple: 2,4,8,10

Mean of tuple: 6.0

**Question 13**

Write a program to check the mode of a tuple is actually an element with maximum occurrences.

**Solution**

tup = eval(input("Enter a tuple: "))

maxCount = 0

mode = 0

for i in tup :

count = tup.count(i)

if maxCount < count:

maxCount = count

mode = i

print("mode:", mode)

**Output**

Enter a tuple: 2,4,5,2,5,2

mode = 2

**Question 14**

Write a program to calculate the average of a tuple's element by calculating its sum and dividing it with the count of the elements. Then compare it with the mean obtained using mean() of statistics module.

**Solution**

import statistics

tup = eval(input("Enter a tuple: "))

tup\_sum = sum(tup)

tup\_len = len(tup)

print("Average of tuple element is:", tup\_sum / tup\_len)

print("Mean of tuple element is:", statistics.mean(tup))

**Output**

Enter a tuple: 2,3,4,5,6,7,8,9,10

Average of tuple element is: 6.0

Mean of tuple element is: 6

**Question 15**

**Mean of means.** Given a nested tuple tup1 = ( (1, 2), (3, 4.15, 5.15), ( 7, 8, 12, 15)). Write a program that displays the means of individual elements of tuple tup1 and then displays the mean of these­ computed means. That is for above tuple, it­ should display as :  
Mean element 1 : 1. 5 ;  
Mean element 2 : 4.1 ;  
Mean element 3 : 10. 5 ;  
Mean of means 5. 366666

**Solution**

tup1 = ((1, 2), (3, 4.15, 5.15), ( 7, 8, 12, 15))

total\_mean = 0

tup1\_len = len(tup1)

for i in range(tup1\_len):

mean = sum(tup1[i]) / len(tup1[i])

print("Mean element", i + 1, ":", mean)

total\_mean = total\_mean + mean

print("Mean of means" ,total\_mean / tup1\_len)

**Output**

Mean element 1 : 1.5

Mean element 2 : 4.1000000000000005

Mean element 3 : 10.5

Mean of means 5.366666666666667