Python Dev Test : Assignment

Question 1:

Write a for loop to iterate through the list A = [1, 2, 3, 4, 5, 6]. Square each element of the list in one-by-one fashion and print them. After the end of the iteration, print - "The sequence has ended".

Code:

```
def sequence():
    """
    Function for squaring each element in a list and printing them
    """

    A = [1, 2, 3, 4, 5, 6]      # Sample List
    for i in range(len(A)):     # 'for' loop to traverse the List
        A[i] = A[i]**2          # Squaring each element of the list
    for a in A:                 # Printing the New list
        print(a)

    print("The sequence has ended")     # Showing that the sequence has ended


if __name__ == '__main__':
    sequence()
```

Input/Output:

```
1
4
9
16
25
36
The sequence has ended
```

Question 2:

If choice of user = 2, print the pattern - >

```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
  _ _
 _ _ _
_ _ _ _
_ _ _ _ _
```

If choice of user = 1, print the pattern - >

```
* * * * *
 * * * *
  * * *
   * *
    *
```

If choice of the user = any_other_choice_other_than_1_and_2, print the message - >

'Invalid Input'

Code:

```
def pattern1():
    """
    Function for displaying Pattern:
    * * * * *
     * * * *
      * * *
       * *
        *
```

```
    """

    for i in range(5, 0, -1):        # Outer loop for the number of rows
        for j in range(5, i, -1):    # Inner loop for Printing the spaces
            print(' ', end='')
        for j in range(i):           # Inner loop for Printing the Printing each row
            print('*', end=' ')
        print()                      # Print statement for changing to new line


def pattern2():
    """
    Function for displaying Pattern:

      _ _
     _ _ _
    _ _ _ _
   _ _ _ _ _

    """

    for i in range(1, 5):            # Outer loop for the number of rows
        for j in range(5-i-1):       # Inner loop for Printing the spaces
            print(' ', end='')
        for j in range(i+1):         # Inner loop for Printing the Printing each row
            print('_', end=' ')
        print()                      # Print statement for changing to new line


if __name__ == '__main__':
    ch = int(input("Enter choice: "))   # Asking the user for the choice
```

```
    if ch == 1:
        pattern1()              # Calling Function pattern1 if choice is 1
    elif ch == 2:
        pattern1()              # Calling function pattern1 and pattern2 if choice is 2
        pattern2()
    else:
        print("Invalid Input")      # Printing 'Invalid Input' if choice is other than 1 or 2
```

Input/Output:

```
Enter choice: 1
* * * * *
 * * * *
  * * *
   * *
    *


Enter choice: 2
* * * * *
 * * * *
  * * *
   * *
    *
    _ _
   _ _ _
  _ _ _ _
 _ _ _ _ _
```

Question 3:

Create a tuple t_1 = (1, 4, 9, 16, 25, 36). Square each element of the tuple using tuple comprehension and store the result in a variable known as t_modified. Find element at index position 4 of the tuple t_modified. Now slice the modified tuple in such a way that the sliced tuple includes only elements from index position 1 to 3 and store this sliced tuple in a variable known as t_sliced.

Code:

```
def tuple_operations():
    """
    Doing some Tuple Operations
    """

    t_1 = (1, 4, 9, 16, 25, 36)
    t_modified = tuple([x**2 for x in t_1])     # Squaring each element of the tuple using list comprehension
                                                # and coverting it to tuple
    t_modified_4th_element = t_modified[4]      # Storing 4th element of the modified tuple
    t_sliced = t_modified[1:4]                  # Slicing the tuple to give 1st to 3rd elements of the tuple

    # Displaying all the Data
    print("t_1: %s" % (t_1,))
    print("t_modified: %s" % (t_modified,))
    print("Element at index position 4 of t_modified: %s" % (t_modified_4th_element,))
    print("t_sliced: %s" % (t_sliced, ))


if __name__ == '__main__':
    tuple_operations()
```

Input/Output:

t_1: (1, 4, 9, 16, 25, 36)

t_modified: (1, 16, 81, 256, 625, 1296)

Element at index position 4 of t_modified: 625

t_sliced: (16, 81, 256)

Question 4:

Show by raising a error how tuple are immutable and also define what exactly immutability is in your own words.

Code:

```
def tuple_immutability_check():
    """
    Showing Immutability of Tuple
    Immutable means unchanging over time or unable to be changed
    In Python, Immutability means that the value of an object cannot be changed once
    it is assigned some value. The value of the object becomes permanent once created.
    Even when passed to a function a copy of the immutable object is passed keeping the
    original same.
    """

    tup = (1, 2, 3, 4, 5)   # Arbitrary Tuple
    try:
        tup[2] = 9            # Changing the value of one of the elements of tuple
    except TypeError:         # Execute this block if TypeError is encountered
        print(" Tuple is of immutable type\n It's value can't be changed\n It doesn't support item assignment")


if __name__ == '__main__':
    tuple_immutability_check()
```

Input/Output:

```
Tuple is of immutable type
 It's value can't be changed
 It doesn't support item assignment
```

Question 5:

Create a frozenset named frozen_set_1 containing the elements: 'A', 'B', 'C' and 'D' and combine it using union with a frozenset named frozen_set_2 containing elements 'A', 2, 'C' and 4. The final combined frozenset must be named frozenset_union. Now find the common elements in frozen_set_1 and frozen_set_2 and store the result in a variable named frozenset_common. Lastly, in a new forzenset named forzenset_difference store the elements of frozen_set_1 which are not in frozen_set_2 and in a new frozenset named frozenset_distinct store the elements which are unique to frozen_set_1 and frozen_set_2.

Code:

```
def frozen_set():
    """
    Doing Operations on Frozen Sets
    """

    frozen_set_1 = frozenset(('A', 'B', 'C', 'D'))   # Creating an arbitrary frozen set
    frozen_set_2 = frozenset(('A', 2, 'C', 4))       # Creating another arbitrary frozen set

    # Frozen Set Operations
    frozenset_union = frozen_set_1.union(frozen_set_2)                  # Combined Frozen Set
    frozenset_common = frozen_set_1.intersection(frozen_set_2)          # Common elements b/w 2 frozen sets
    frozenset_difference = frozen_set_1.difference(frozen_set_2)        # Elements in Frozen Set 1 and not in 2
    frozenset_distinct = frozen_set_1.symmetric_difference(frozen_set_2)    # Elements Distinct to Frozen Set 1 and 2

    # Displaying all the Frozen Sets
    print("frozen_set_1: %s" % frozen_set_1)
    print("frozen_set_2: %s" % frozen_set_2)
    print("frozenset_union: %s" % frozenset_union)
    print("frozenset_common: %s" % frozenset_common)
```

```
    print("frozenset_difference: %s" % frozenset_difference)
    print("frozenset_distinct: %s" % frozenset_distinct)


if __name__ == '__main__':
    frozen_set()
```

Input/Output:

```
frozen_set_1: frozenset({'C', 'A', 'B', 'D'})
frozen_set_2: frozenset({'C', 2, 'A', 4})
frozenset_union: frozenset({2, 'A', 4, 'D', 'C', 'B'})
frozenset_common: frozenset({'C', 'A'})
frozenset_difference: frozenset({'B', 'D'})
frozenset_distinct: frozenset({2, 4, 'D', 'B'})
```

Question 6:

Write a python program to remove items in a list containing the character 'a' or 'A'. Use lambda function for it. For this program pass in as argument the list: list_a = ["car", "place", "tree", "under", "grass", "price"] to the lambda function named remove_items_containing_a_or_A.

Code:

```
def main():
    """
    Function to elements containing 'a' 'A' in the list
    Also used filter function along with lambda function to remove the words
    """

    list_a = ["car", "place", "tree", "under", "grass", "price"]    # Arbitrary list
    remove_items_containing_a_or_A = list(filter(lambda word: 'a' not in word and 'A' not in word, list_a))
    print(remove_items_containing_a_or_A)


if __name__ == '__main__':
    main()
```

Input/Output:

```
['tree', 'under', 'price']
```

Question 7:

Create a custom exception class which can handle "IndexError" as well as "ValueError" such that it can display its own custom error message when we use index which is not valid in a list. Take list as list_a = [1, 2, 3, 4, 5].

Code:

```
# Base Exception Class
class Error(IndexError, ValueError):
    pass



# If the index is out of range
class IndexOutOfRangeError(Error):
    def __init__(self, index, start, end):
        self.index = int(index)
        self.start = start
        self.end = end
        # Combining the error message
        self.msg = "The index {} is incorrect and index should lie between {} and {}.".format(self.index, self.start,  self.end)



# If the Entered Index is Not Integer
class NonIntegerValueError(Error):
    def __init__(self):
        self.msg = "Use an Integer value as the input."


def main():
    try:
        list_a = [1, 2, 3, 4, 5]            # Take an arbitrary list
        start = -(len(list_a))              # Starting negative index
```

```
        end = len(list_a) - 1                # Ending Positive index
        index = input("Enter the index = ")     # Taking index as input from the user
        # Declaring objects of Custom Exception Classes
        error1 = NonIntegerValueError()
        error2 = IndexOutOfRangeError(index, start, end)
        # If the Index is not integer
        if not index.isdigit():
            raise NonIntegerValueError()
        # If the index is not in appropriate range
        elif int(index) > 4 and int(index) < -5:
            raise IndexOutOfRangeError(index, start, end)
        # Printing the Element is no Exception encountered
        print("Element at Index {} is {}".format(index, list_a[int(index)]))
    except IndexError:
        print(error2.msg)
    except ValueError:
        print(error1.msg)


if __name__ == '__main__':
    main()
```

Input/Output:

```
Enter the index = 10
The index 10 is incorrect and index should lie between -5 and 4.

Enter the index = abc
Use an Integer value as the input.

Enter the index = 3
Element at Index 3 is 4
```