**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