PYTHON SUBJECTIVE ANSWERS

Ans1

```
def change_char(str1):
  char = str1[0]
  str1 = str1.replace(char, '$')
  str1 = char + str1[1:]

  return str1

print(change_char('prospect'))
```

Ans2

```
def chars_mix_up(a, b):

  new_a = b[:2] + a[2:]

  new_b = a[:2] + b[2:]



  return new_a + ' ' + new_b

print(chars_mix_up('abc', 'xyz'))
```

Ans 3

```
def add_string(str1):
  length = len(str1)

  if length > 2:
    if str1[-3:] == 'ing':
      str1 += 'ly'
    else:
      str1 += 'ing'

  return str1
print(add_string('ab'))
print(add_string('abc'))
print(add_string('string'))
```

Ans 4

```
def not_poor(str1):
  snot = str1.find('not')
  spoor = str1.find('poor')


  if spoor > snot and snot>0 and spoor>0:
    str1 = str1.replace(str1[snot:(spoor+4)], 'good')
    return str1
  else:
    return str1
print(not_poor('The lyrics is not that poor!'))
print(not_poor('The lyrics is poor!'))
```

Ans 5

```
def odd_values_string(str):
  result = ""
  for i in range(len(str)):
    if i % 2 == 0:
      result = result + str[i]
  return result

print(odd_values_string('abcdef'))
print(odd_values_string('python'))
```

Ans 6

```
def insert_end(str):

       sub_str = str[-2:]

       return sub_str * 4



print(insert_end('Python'))

print(insert_end('Exercises'))
```

Ans 7

```
def first_three(str):

    return str[:3] if len(str) > 3 else str



print(first_three('ipy'))

print(first_three('python'))

print(first_three('py'))
```

Copy
Sample Output:

Ans 8

```
x = 3.1415926

y = 12.9999

print("\nOriginal Number: ", x)

print("Formatted Number: "+"{:.2f}".format(x));

print("Original Number: ", y)

print("Formatted Number: "+"{:.2f}".format(y));

print()
```

Ans 9

```
x = 0.25
y = -0.25
print("\nOriginal Number: ", x)
print("Formatted Number with percentage: "+"{:.2%}".format(x));
print("Original Number: ", y)
print("Formatted Number with percentage: "+"{:.2%}".format(y));
print()
```

Sample Output:

```
Original Number:  0.25
Formatted Number with percentage: 25.00%
```

```
Original Number:  -0.25
Formatted Number with percentage: -25.00%
```

Ans 10

```
str1 = 'The quick brown fox jumps over the lazy dog.'

print()

print(str1.count("fox"))

print()
```

Ans 11

```
import collections
str1 = 'thequickbrownfoxjumpsoverthelazydog'
d = collections.defaultdict(int)
for c in str1:
    d[c] += 1

for c in sorted(d, key=d.get, reverse=True):
  if d[c] > 1:
      print('%s %d' % (c, d[c]))
```

Ans12

```
area = 1256.66
volume = 1254.725
decimals = 2
print("The area of the rectangle is {0:.{1}f}cm\u00b2".format(area, decimals))
decimals = 3
print("The volume of the cylinder is {0:.{1}f}cm\u00b3".format(volume,
decimals))
```

Ans 13

```
import string
alphabet = set(string.ascii_lowercase)
input_string = 'The quick brown fox jumps over the lazy dog'
print(set(input_string.lower()) >= alphabet)
input_string = 'The quick brown fox jumps over the lazy cat'
print(set(input_string.lower()) >= alphabet)
```

Ans 14

```
def word_count(str):
    counts = dict()
    words = str.split()

    for word in words:
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

    counts_x = sorted(counts.items(), key=lambda kv: kv[1])
    #print(counts_x)
    return counts_x[-2]

print(word_count("Both of these issues are fixed by postponing the evaluation of annotations. Instead of compiling code which executes expressions in annotations at their definition time, the compiler stores the annotation in a string form equivalent to the AST of the expression in question. If needed, annotations can be resolved at runtime using typing.get_type_hints(). In the common case where this is not required, the annotations are cheaper to store (since short strings are interned by the interpreter) and make startup time faster."))
```

Ans 15

```
import collections
def min_window(str1, str2):
    result_char, missing_char = collections.Counter(str2), len(str2)
    i = p = q = 0
    for j, c in enumerate(str1, 1):
        missing_char -= result_char[c] > 0
        result_char[c] -= 1
        if not missing_char:
            while i < q and result_char[str1[i]] < 0:
                result_char[str1[i]] += 1
                i += 1
            if not q or j - i <= q - p:
                p, q = i, j
    return str1[p:q]

str1 = "PRWSOERIUSFK"
str2 = "OSU"
```

```
print("Original Strings:\n",str1,"\n",str2)
print("Minimum window:")
print(min_window(str1,str2))
```

Ans 16

```
from collections import defaultdict


def find_sub_string(str):

    str_len = len(str)


    # Count all distinct characters.

    dist_count_char = len(set([x for x in str]))


    ctr, start_pos, start_pos_index, min_len = 0, 0, -1, 9999999999

    curr_count = defaultdict(lambda: 0)

    for i in range(str_len):

        curr_count[str[i]] += 1


        if curr_count[str[i]] == 1:

            ctr += 1


        if ctr == dist_count_char:

            while curr_count[str[start_pos]] > 1:

                if curr_count[str[start_pos]] > 1:

                    curr_count[str[start_pos]] -= 1

                start_pos += 1


            len_window = i - start_pos + 1

            if min_len > len_window:
```

```
            min_len = len_window

            start_pos_index = start_pos

    return str[start_pos_index: start_pos_index + min_len]


str1 = "asdaewsqgtwwsa"

print("Original Strings:\n",str1)

print("\nSmallest window that contains all characters of the said string:")

print(find_sub_string(str1))
```

Ans 17

```
def count_k_dist(str1, k):
        str_len = len(str1)

        result = 0

        ctr = [0] * 27

        for i in range(0, str_len):
                dist_ctr = 0

                ctr = [0] * 27

                for j in range(i, str_len):

                        if(ctr[ord(str1[j]) - 97] == 0):
                                dist_ctr += 1

                        ctr[ord(str1[j]) - 97] += 1

                        if(dist_ctr == k):
                                result += 1
                        if(dist_ctr > k):
                                break

        return result

str1 = input("Input a string (lowercase alphabets):")
k = int(input("Input k: "))
```

```
print("Number of substrings with exactly", k, "distinct characters : ", end = "")
print(count_k_dist(str1, k))
```

Ans 18.

```
def number_of_substrings(str):
        str_len = len(str);
        return int(str_len * (str_len + 1) / 2);

str1 = input("Input a string: ")
print("Number of substrings:")
print(number_of_substrings(str1))
```

Ans 19

```
def no_of_substring_with_equalEnds(str1):
        result = 0;
        n = len(str1);
        for i in range(n):
                for j in range(i, n):
                        if (str1[i] == str1[j]):
                                result = result + 1
        return result
str1 = input("Input a string: ")
print(no_of_substring_with_equalEnds(str1))
```

Ans 20.

```
def match_words(words):
  ctr = 0

  for word in words:
    if len(word) > 1 and word[0] == word[-1]:
      ctr += 1
  return ctr

print(match_words(['abc', 'xyz', 'aba', '1221']))
```

Ans 21.

```
def last(n): return n[-1]

def sort_list_last(tuples):
  return sorted(tuples, key=last)
```

```
print(sort_list_last([(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]))
```

Ans 22

```
a = [10,20,30,20,10,50,60,40,80,50,40]



dup_items = set()

uniq_items = []

for x in a:

    if x not in dup_items:

        uniq_items.append(x)

        dup_items.add(x)



print(dup_items)
```

Ans 23

```
def long_words(n, str):
    word_len = []
    txt = str.split(" ")
    for x in txt:
        if len(x) > n:
            word_len.append(x)
    return word_len
print(long_words(3, "The quick brown fox jumps over the lazy dog"))
```

Ans 24

```
color = ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']

color = [x for (i,x) in enumerate(color) if i not in (0,4,5)]

print(color)
```

Ans 25.

```
import itertools
print(list(itertools.permutations([1,2,3])))
```

Ans 26.

```
L = [(1, 2), (3, 4), (1, 2), (5, 6), (7, 8), (1, 2), (3, 4), (3, 4),

 (7, 8), (9, 10)]

print("Original List: ", L)

print("Sorted Unique Data:",sorted(set().union(*L)))
```

Ans 27.

```
class py_solution:
    def int_to_Roman(self, num):
        val = [
            1000, 900, 500, 400,
            100, 90, 50, 40,
            10, 9, 5, 4,
            1
            ]
        syb = [
            "M", "CM", "D", "CD",
            "C", "XC", "L", "XL",
            "X", "IX", "V", "IV",
            "I"
            ]
        roman_num = ''
        i = 0
        while  num > 0:
            for _ in range(num // val[i]):
                roman_num += syb[i]
                num -= val[i]
            i += 1
        return roman_num


print(py_solution().int_to_Roman(1))

print(py_solution().int_to_Roman(4000))
```

Ans 28.

```
class py_solution:
    def roman_to_int(self, s):
        rom_val = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
        int_val = 0
        for i in range(len(s)):
            if i > 0 and rom_val[s[i]] > rom_val[s[i - 1]]:
                int_val += rom_val[s[i]] - 2 * rom_val[s[i - 1]]
            else:
                int_val += rom_val[s[i]]
        return int_val

print(py_solution().roman_to_int('MMMCMLXXXVI'))
print(py_solution().roman_to_int('MMMM'))
print(py_solution().roman_to_int('C'))
```

Ans 29

```
class py_solution:

   def is_valid_parenthese(self, str1):

        stack, pchar = [], {"(": ")", "{": "}", "[": "]"}

        for parenthese in str1:

            if parenthese in pchar:

                stack.append(parenthese)

            elif len(stack) == 0 or pchar[stack.pop()] != parenthese:

                return False

        return len(stack) == 0



print(py_solution().is_valid_parenthese("(){}[]"))

print(py_solution().is_valid_parenthese("()[{)}"))

print(py_solution().is_valid_parenthese("()"))
```

Ans 30

```
class py_solution:
    def sub_sets(self, sset):
        return self.subsetsRecur([], sorted(sset))
```

```
   def subsetsRecur(self, current, sset):
       if sset:
           return self.subsetsRecur(current, sset[1:]) + self.subsetsRecur(current +
[sset[0]], sset[1:])
       return [current]

print(py_solution().sub_sets([4,5,6]))
```

Ans 31.

```
class py_solution:

  def twoSum(self, nums, target):

       lookup = {}

       for i, num in enumerate(nums):

           if target - num in lookup:

               return (lookup[target - num], i )

           lookup[num] = i

print("index1=%d, index2=%d" %
py_solution().twoSum((10,20,10,40,50,60,70),50))
```

Ans 32.

```
class py_solution:

 def threeSum(self, nums):

       nums, result, i = sorted(nums), [], 0

       while i < len(nums) - 2:

           j, k = i + 1, len(nums) - 1

           while j < k:

               if nums[i] + nums[j] + nums[k] < 0:

                   j += 1

               elif nums[i] + nums[j] + nums[k] > 0:
                   k -= 1

               else:

                   result.append([nums[i], nums[j], nums[k]])

                   j, k = j + 1, k - 1

                   while j < k and nums[j] == nums[j - 1]:

                       j += 1

                   while j < k and nums[k] == nums[k + 1]:

                       k -= 1

           i += 1

           while i < len(nums) - 2 and nums[i] == nums[i - 1]:

               i += 1

       return result



print(py_solution().threeSum([-25, -10, -7, -3, 2, 4, 8, 10]))
```

Ans 33.

```
class py_solution:
   def pow(self, x, n):
        if x==0 or x==1 or n==1:
            return x

        if x==-1:
            if n%2 ==0:
                return 1
            else:
                return -1
        if n==0:
            return 1
        if n<0:
            return 1/self.pow(x,-n)
        val = self.pow(x,n//2)
        if n%2 ==0:
            return val*val
        return val*val*x

print(py_solution().pow(2, -3));
print(py_solution().pow(3, 5));
print(py_solution().pow(100, 0));
```

Ans 34

```
class IOString():
    def __init__(self):
        self.str1 = ""

    def get_String(self):
        self.str1 = input()

    def print_String(self):
        print(self.str1.upper())

str1 = IOString()
str1.get_String()
str1.print_String()
```

Ans 35

```
class Rectangle():

    def __init__(self, l, w):

        self.length = l

        self.width  = w



    def rectangle_area(self):

        return self.length*self.width



newRectangle = Rectangle(12, 10)

print(newRectangle.rectangle_area())
```

Ans 36.

```
 class Circle():

    def __init__(self, r):
        self.radius = r

    def area(self):
        return self.radius**2*3.14

    def perimeter(self):
        return 2*self.radius*3.14

NewCircle = Circle(8)
print(NewCircle.area())
print(NewCircle.perimeter())
```

Ans 37.

```
import itertools

x = itertools.cycle('ABCD')

print(type(x).__name__)
```

Ans 38.

```
from collections import Counter
classes = (
    ('V', 1),
    ('VI', 1),
    ('V', 2),
    ('VI', 2),
    ('VI', 3),
    ('VII', 1),
)
students = Counter(class_name for class_name, no_students in classes)
print(students)
```

Ans 39.

```
from collections import OrderedDict
                        dict = {'Afghanistan': 93, 'Albania': 355, 'Algeria': 213, 'Andorra':
                        376, 'Angola': 244}
```

```
new_dict = OrderedDict(dict.items())
for key in new_dict:
    print (key, new_dict[key])

print("\nIn reverse order:")
for key in reversed(new_dict):
    print (key, new_dict[key])
```

Ans 40.

```
from collections import Counter

def compare_lists(x, y):
    return Counter(x) == Counter(y)
n1 = [20, 10, 30, 10, 20, 30]
n2 = [30, 20, 10, 30, 20, 50]
print(compare_lists(n1, n2))
```

Ans 41.

```
from array import array

a = array("I", (12,25))

print("Array buffer start address in memory and number of elements.")

print(a.buffer_info())
```

Ans 42.

```
import array

import binascii

a = array.array('i', [1,2,3,4,5,6])

print("Original array:")

print('A1:', a)

bytes_array = a.tobytes()

print('Array of bytes:', binascii.hexlify(bytes_array))
```

Ans 43

```
from array import array
import binascii
array1 = array('i', [7, 8, 9, 10])
print('array1:', array1)
as_bytes = array1.tobytes()
print('Bytes:', binascii.hexlify(as_bytes))
array2 = array('i')
array2.frombytes(as_bytes)
print('array2:', array2)
```

Ans 44

```
import heapq
heap = []
heapq.heappush(heap, ('V', 3))
heapq.heappush(heap, ('V', 2))
heapq.heappush(heap, ('V', 1))
print("Items in the heap:")
for a in heap:
        print(a)
print("----------------------")
print("The smallest item in the heap:")
print(heap[0])
print("----------------------")
print("Pop the smallest item in the heap:")
heapq.heappop(heap)
for a in heap:
    print(a)
```

Ans 45.

```
import bisect

def index(a, x):

  i = bisect.bisect_left(a, x)

  return i



a = [1,2,4,5]
```

```
print(index(a, 6))

print(index(a, 3))
```

Ans 46

```
mport queue

q = queue.Queue()

#insert items at the end of the queue

for x in range(4):

    q.put(str(x))

#remove items from the head of the queue

while not q.empty():

    print(q.get(), end=" ")

print("\n")
```

Ans 47

```
def harmonic_sum(n):

  if n < 2:

    return 1

  else:

    return 1 / n + (harmonic_sum(n - 1))



print(harmonic_sum(7))

print(harmonic_sum(4))
```

Ans 48.

```
import numpy as np
x = np.ones((5,5))
print("Original array:")
print(x)
print("1 on the border and 0 inside in the array")
x[1:-1,1:-1] = 0
print(x)
```

Ans 49.

```
import numpy as np

x = np.ones((3,3))

print("Checkerboard pattern:")

x = np.zeros((8,8),dtype=int)

x[1::2,::2] = 1

x[::2,1::2] = 1

print(x)
```

Ans 50.

```
import numpy as np
# Create an empty array
x = np.empty((3,4))
print(x)
# Create a full array
y = np.full((3,3),6)
print(y)
```

51.

```
import numpy as np
fvalues = [0, 12, 45.21, 34, 99.91]
F = np.array(fvalues)
print("Values in Fahrenheit degrees:")
print(F)
print("Values in  Centigrade degrees:")
print(5*F/9 - 5*32/9)
```

Ans 52.

```
import numpy as np
x = np.sqrt([1+0j])
y = np.sqrt([0+1j])
print("Original array:x ",x)
print("Original array:y ",y)
print("Real part of the array:")
print(x.real)
print(y.real)
print("Imaginary part of the array:")
print(x.imag)
print(y.imag)
```

Ans 53.

```
import numpy as np
array1 = np.array([0, 10, 20, 40, 60])
print("Array1: ",array1)
array2 = [0, 40]
print("Array2: ",array2)
print("Compare each element of array1 and array2")
print(np.in1d(array1, array2))
```

Ans 54.

```
import numpy as np
array1 = np.array([0, 10, 20, 40, 60])
print("Array1: ",array1)
array2 = [10, 30, 40]
print("Array2: ",array2)
print("Common values between two arrays:")
print(np.intersect1d(array1, array2))
```

Ans 55.

```
import numpy as np
x = np.array([10, 10, 20, 20, 30, 30])
print("Original array:")
print(x)
print("Unique elements of the above array:")
print(np.unique(x))
x = np.array([[1, 1], [2, 3]])
print("Original array:")
print(x)
print("Unique elements of the above array:")
print(np.unique(x))
```

Ans 56.

```
import numpy as np
array1 = np.array([0, 10, 20, 40, 60, 80])
```

```
print("Array1: ",array1)
array2 = [10, 30, 40, 50, 70]
print("Array2: ",array2)
print("Unique values that are in only one (not both) of the input arrays:")
print(np.setxor1d(array1, array2))
```

Ans 57.

```
import numpy as np
print(np.all([[True,False],[True,True]]))
print(np.all([[True,True],[True,True]]))
print(np.all([10, 20, 0, -50]))
print(np.all([10, 20, -50]))
```

Ans 58.

```
import numpy as np

print(np.any([[False,False],[False,False]]))

print(np.any([[True,True],[True,True]]))

print(np.any([10, 20, 0, -50]))

print(np.any([10, 20, -50]))
```

59. import numpy as np

```
a = [1, 2, 3, 4]
print("Original array")
print(a)
print("Repeating 2 times")
x = np.tile(a, 2)
print(x)
print("Repeating 3 times")
x = np.tile(a, 3)
print(x)
```

Ans 60.

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6])

print("Original array: ",x)

print("Maximum Values: ",np.argmax(x))
print("Minimum Values: ",np.argmin(x))
```

Ans 61.

```
import numpy as np
a = np.array([1, 2])
b = np.array([4, 5])
print("Array a: ",a)
print("Array b: ",b)
print("a > b")
print(np.greater(a, b))
print("a >= b")
print(np.greater_equal(a, b))
print("a < b")
print(np.less(a, b))
print("a <= b")
print(np.less_equal(a, b))
```

Ans 62.

```
import numpy as np

a = np.array([[4, 6],[2, 1]])

print("Original array: ")

print(a)

print("Sort along the first axis: ")

x = np.sort(a, axis=0)

print(x)

print("Sort along the last axis: ")

y = np.sort(x, axis=1)

print(y)
```

Ans 63.

```
import numpy as np

first_names =    ('Margery', 'Betsey', 'Shelley', 'Lanell', 'Genesis')

last_names = ('Woolum', 'Battle', 'Plotner', 'Brien', 'Stahl')
```

```
x = np.lexsort((first_names, last_names))

print(x)
```

Ans 64.

```
import numpy as np
x = np.array([[0, 10, 20], [20, 30, 40]])
print("Original array: ")
print(x)
print("Values bigger than 10 =", x[x>10])
print("Their indices are ", np.nonzero(x > 10))
```

Ans 65.

```
import numpy as np

n = np.zeros((4,4))

print("%d bytes" % (n.size * n.itemsize))
```

Ans 66

```
import numpy as np
print("Create an array of zeros")
x = np.zeros((1,2))
print("Default type is float")
print(x)
print("Type changes to int")
x = np.zeros((1,2), dtype = np.int)
print(x)
print("Create an array of ones")
y= np.ones((1,2))
print("Default type is float")
print(y)
print("Type changes to int")
y = np.ones((1,2), dtype = np.int)
print(y)
```

Ans 67.

```
import numpy as np

x = np.array([1, 2, 3, 4, 5, 6])
print("6 rows and 0 columns")
print(x.shape)
```

```
y = np.array([[1, 2, 3],[4, 5, 6],[7,8,9]])
print("(3, 3) -> 3 rows and 3 columns ")
print(y)

x = np.array([1,2,3,4,5,6,7,8,9])
print("Change array shape to (3, 3) -> 3 rows and 3 columns ")
x.shape = (3, 3)
print(x)
```

Ans 68.

```
import numpy as np
x = np.array([1, 2, 3, 4, 5, 6])
y = np.reshape(x,(3,2))
print("Reshape 3x2:")
print(y)
z = np.reshape(x,(2,3))
print("Reshape 2x3:")
print(z)
```

Ans 69.

```
import numpy as np

#using no.full

x = np.full((3, 5), 2, dtype=np.uint)

print(x)

#using no.ones

y = np.ones([3, 5], dtype=np.uint) *2

print(y)
```

Ans 70.

```
import numpy as np
x = np.eye(3)
print(x)
```

Ans 71.

```
import numpy as np
x = np.arange(1, 15)
```

```
print("Original array:",x)
print("After splitting:")
print(np.split(x, [2, 6]))
```

Ans 72

```
import numpy as np
x = np.arange(16).reshape((4, 4))
print("Original array:",x)
print("After splitting horizontally:")
print(np.hsplit(x, [2, 6]))
```

Ans 73.

```
import numpy as np
x = np.zeros((5,5))
print("Original array:")
print(x)
print("Row values ranging from 0 to 4.")
x += np.arange(5)
print(x)
```

Ans 74.

```
import numpy as np
x = np.zeros((3,), dtype=('i4,f4,a40'))
new_data = [(1, 2., "Albert Einstein"), (2, 2., "Edmond Halley"), (3, 3., "Gertrude
B. Elion")]
x[:] = new_data
print(x)
```

Ans 75.

```
import numpy as np
x = np.array([-1, -4, 0, 2, 3, 4, 5, -6])
print("Original array:")
print(x)
print("Replace the negative values of the said array with 0:")
x[x < 0] = 0
print(x)
```

Ans 76.

```
import numpy as np

import matplotlib.pyplot as plt
```

```
plt.hist([1, 2, 1], bins=[0, 1, 2, 3, 5])

plt.show()
```

Ans 77.

```
import numpy as np

import matplotlib.pyplot as plt

arr = np.random.randint(1, 50, 10)

y, x = np.histogram(arr, bins=np.arange(51))

fig, ax = plt.subplots()

ax.plot(x[:-1], y)

fig.show()
```

Ans 78.

```
import numpy as np

arra_data = np.arange(0,16).reshape((4, 4))
print("Original array:")
print(arra_data)
print("\nExtracted data: Second row")
print(arra_data[1,:])
```

Ans 79.

```
import numpy as np
arra_data = np.arange(0,16).reshape((4, 4))
print("Original array:")
print(arra_data)
print("\nExtracted data: First element of the second row and fourth element of
fourth row  ")
print(arra_data[[1,3], [0,3]])
```

Ans 80.

```
import numpy as np

A = np.ones((3,3))
B = np.arange(3)
```

```
print("Original array:")
print("Array-1")
print(A)
print("Array-2")
print(B)
print("A + B:")
new_array = A + B
print(new_array)
```

Ans 81.

```
import numpy as np
x = np.array([24, 27, 30, 29, 18, 14])
print("Original array:")
print(x)
y = np.empty_like (x)
y[:] = x
print("\nCopy of the said array:")
print(y)
```

Ans 82.

```
import numpy as np

num = np.arange(36)

arr1 = np.reshape(num, [4, 9])

print("Original array:")

print(arr1)

result  = arr1.sum(axis=0)

print("\nSum of all columns:")

print(result)
```

Ans 83.

```
import numpy as np
arr1 = np.array([[10, 20 ,30], [40, 50, np.nan], [np.nan, 6, np.nan], [np.nan, np.nan,
np.nan]])
print("Original array:")
print(arr1)
temp = np.ma.masked_array(arr1,np.isnan(arr1))
result = np.mean(temp, axis=1)
print("Averages without NaNs along the said array:")
print(result.filled(np.nan))
```

Ans 84.

```
import numpy as np

x = np.array([10,-10,10,-10,-10,10])

y = np.array([.85,.45,.9,.8,.12,.6])

print("Original arrays:")

print(x)

print(y)

result = np.sum((x == 10) & (y > .5))

print("\nNumber of instances of a value occurring in one aray on the condition of
another array:")

print(result)
```

Ans 85.

```
import numpy as np
from ast import literal_eval
udict = """{"column0":{"a":1,"b":0.0,"c":0.0,"d":2.0},
  "column1":{"a":3.0,"b":1,"c":0.0,"d":-1.0},
  "column2":{"a":4,"b":1,"c":5.0,"d":-1.0},
  "column3":{"a":3.0,"b":-1.0,"c":-1.0,"d":-1.0}
 }"""
t = literal_eval(udict)
print("\nOriginal dictionary:")
print(t)
print("Type: ",type(t))
result_nparra = np.array([[v[j] for j in ['a', 'b', 'c', 'd']] for k, v in t.items()])
print("\nndarray:")
print(result_nparra)
print("Type: ",type(result_nparra))
```

Ans 86.

```
import numpy as np
arra = np.array([[ 1,  1,  0],
                 [ 0,  0,  0],
                 [ 0,  2,  3],
                 [ 0,  0,  0],
                 [ 0, -1,  1],
                 [ 0,  0,  0]])

print("Original array:")
print(arra)
temp = {(0, 0, 0)}
result = []
for idx, row in enumerate(map(tuple, arra)):
    if row not in temp:
        result.append(idx)
print("\nNon-zero unique rows:")
print(arra[result])
```

Ans 87.

```
import numpy as np

x = np.array([1+2j,3+4j])

print("First array:")

print(x)

y = np.array([5+6j,7+8j])

print("Second array:")

print(y)

z = np.vdot(x, y)

print("Product of above two arrays:")

print(z)
```

Ans 88.

```
import numpy as np
x = [[1, 0], [1, 1]]
y = [[3, 1], [2, 2]]
print("Matrices and vectors.")
print("x:")
print(x)
print("y:")
print(y)
print("Matrix product of above two arrays:")
print(np.matmul(x, y))
```

Ans 89.

```
import numpy as np

print("Roots of the first polynomial:")

print(np.roots([1, -2, 1]))

print("Roots of the second polynomial:")

print(np.roots([1, -12, 10, 7, -10]))
```

Ans 90.

```
import numpy as np

x = np.array([-1., 0, 1.])
print("Inverse sine:", np.arcsin(x))
print("Inverse cosine:", np.arccos(x))
print("Inverse tangent:", np.arctan(x))
```

Ans 91.

```
import numpy as np

x = np.array([1, 3, 5, 7, 0])
print("Original array: ")
print(x)
print("Difference between neighboring elements, element-wise of the said array.")
print(np.diff(x))
```

Ans 92.

```
import numpy as np

a = np.arange(4).reshape((2,2))

print("Original flattened array:")

print(a)

print("Maximum value of the above flattened array:")

print(np.amax(a))

print("Minimum value of the above flattened array:")

print(np.amin(a))
```

Ans 93.

```
import numpy as np

x = np.arange(12).reshape((2, 6))
print("\nOriginal array:")
print(x)
r1 = np.ptp(x, 1)
r2 = np.amax(x, 1) - np.amin(x, 1)
assert np.allclose(r1, r2)
print("\nDifference between the maximum and the minimum values of the said
array:")
print(r1)
```

Ans 94.

```
import numpy as np

x = np.arange(5)

print("\nOriginal array:")
print(x)
weights = np.arange(1, 6)
r1 = np.average(x, weights=weights)
r2 = (x*(weights/weights.sum())).sum()
assert np.allclose(r1, r2)
print("\nWeighted average of the said array:")
print(r1)
```

Ans 95.

```
import numpy as np
x = np.arange(6)
print("\nOriginal array:")
print(x)
r1 = np.mean(x)
r2 = np.average(x)
assert np.allclose(r1, r2)
print("\nMean: ", r1)
r1 = np.std(x)
r2 = np.sqrt(np.mean((x - np.mean(x)) ** 2 ))
assert np.allclose(r1, r2)
print("\nstd: ", 1)
r1= np.var(x)
r2 = np.mean((x - np.mean(x)) ** 2 )
assert np.allclose(r1, r2)
print("\nvariance: ", r1)
```

Ans 96.

```
import numpy as np

x = np.array([0, 1, 2])

y = np.array([2, 1, 0])

print("\nOriginal array1:")

print(x)

print("\nOriginal array1:")

print(y)

print("\nCovariance matrix of the said arrays:\n",np.cov(x, y))
```

Ans 97.

```
import numpy as np

x = np.array([0, 1, 3])
y = np.array([2, 4, 5])
```

```
print("\nOriginal array1:")
print(x)
print("\nOriginal array1:")
print(y)
print("\nCross-correlation of the said arrays:\n",np.cov(x, y))
```

Ans 98.

```
import numpy as np
x = np.array([0, 1, 3])
y = np.array([2, 4, 5])
print("\nOriginal array1:")
print(x)
print("\nOriginal array1:")
print(y)
print("\nPearson product-moment correlation coefficients of the said
arrays:\n",np.corrcoef(x, y))
```

Ans 99.

```
import numpy as np

array1 = [0, 1, 6, 1, 4, 1, 2, 2, 7]

print("Original array:")

print(array1)

print("Number of occurrences of each value in array: ")

print(np.bincount(array1))
```

Ans 100.

```
import numpy as np
import matplotlib.pyplot as plt
nums = np.array([0.5, 0.7, 1.0, 1.2, 1.3, 2.1])
bins = np.array([0, 1, 2, 3])
print("nums: ",nums)
print("bins: ",bins)
print("Result:", np.histogram(nums, bins))
plt.hist(nums, bins=bins)
plt.show()
```

Ans 101.

```
import pandas as pd
ds1 = pd.Series([2, 4, 6, 8, 10])
ds2 = pd.Series([1, 3, 5, 7, 9])
ds = ds1 + ds2
print("Add two Series:")
print(ds)
print("Subtract two Series:")
ds = ds1 - ds2
print(ds)
print("Multiply two Series:")
ds = ds1 * ds2
print(ds)
print("Divide Series1 by Series2:")
ds = ds1 / ds2
print(ds)
```

Ans 102.

```
import pandas as pd
d1 = {'a': 100, 'b': 200, 'c':300, 'd':400, 'e':800}
print("Original dictionary:")
print(d1)
new_series = pd.Series(d1)
print("Converted series:")
print(new_series)
```

Ans 103.

```
import pandas as pd
s1 = pd.Series(['100', '200', 'python', '300.12', '400'])
print("Original Data Series:")
print(s1)
print("Change the said data type to numeric:")
s2 = pd.to_numeric(s1, errors='coerce')
print(s2)
```

Ans 104.

```
import pandas as pd
d = {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0], 'col3': [7, 5, 8, 12, 1,11]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
s1 = df.ix[:,0]
```

```
print("\n1st column as a Series:")
print(s1)
print(type(s1))
```

Ans 105.

```
import pandas as pd
s = pd.Series(data = [1,2,3,4,5,6,7,8,9,5,3])
print("Original Data Series:")
print(s)
print("Mean of the said Data Series:")
print(s.mean())
print("Standard deviation of the said Data Series:")
print(s.std())
```

Ans 106.

```
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86],
'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

Ans 107.

```
import pandas as pd
import numpy as np

exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
'Matthew', 'Laura', 'Kevin', 'Jonas'],
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data , index=labels)
print("First three rows of the data frame:")
print(df.iloc[:3])
```

Ans 108.

```
import pandas as pd

import numpy as np
```

```
exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
'Matthew', 'Laura', 'Kevin', 'Jonas'],

        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']



df = pd.DataFrame(exam_data , index=labels)

print("Select specific columns and rows:")

print(df.iloc[[1, 3, 5, 6], [1, 3]])
```

Ans 109.

```
import pandas as pd
import numpy as np
exam_data  = {'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily', 'Michael',
'Matthew', 'Laura', 'Kevin', 'Jonas'],
        'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],
        'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
        'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']}
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

df = pd.DataFrame(exam_data , index=labels)
print("\nMean score for each different student in data frame:")
print(df['score'].mean())
```

Ans 110.

```
import pandas as pd
d = {'col1': [1, 2, 3], 'col2': [4, 5, 6], 'col3': [7, 8, 9]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
df.columns = ['Column1', 'Column2', 'Column3']
df = df.rename(columns={'col1': 'Column1', 'col2': 'Column2', 'col3': 'Column3'})
print("New DataFrame after renaming columns:")
print(df)
```

Ans 111.

```
import pandas as pd
df1 = pd.DataFrame({'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',
'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],
'city': ['California', 'Los Angeles', 'California', 'California', 'California', 'Los
Angeles', 'Los Angeles', 'Georgia', 'Georgia', 'Los Angeles']})
g1 = df1.groupby(["city"]).size().reset_index(name='Number of people')
print(g1)
```

Ans 112.

```
import pandas as pd
import numpy as np
d = {'col1': [1, 4, 3, 4, 5], 'col2': [4, 5, 6, 7, 8], 'col3': [7, 8, 9, 0, 1]}
df = pd.DataFrame(data=d)
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
print("Original DataFrame")
print(df)
```

Ans 113.

```
import pandas as pd
import numpy as np
s = pd.Series(['3/11/2000', '3/12/2000', '3/13/2000'])
print("String Date:")
print(s)
r = pd.to_datetime(pd.Series(s))
df = pd.DataFrame(r)
print("Original DataFrame (string to datetime):")
print(df)
```

Ans 114.

```
import pandas as pd

import numpy as np

df = pd.DataFrame()

data = pd.DataFrame({"col1": range(3),"col2": range(3)})
```

```
print("After appending some data:")

df = df.append(data)

print(df)
```

Ans 115.

```
import pandas as pd

d = {'col1': [1, 2, 3, 4, 7], 'col2': [4, 5, 6, 9, 5], 'col3': [7, 8, 12, 1, 11]}

df = pd.DataFrame(data=d)

print("Original DataFrame")

print(df)

print("\nNumber of columns:")

print(len(df.columns))
```

Ans 116.

```
import pandas as pd
d = {'col1': [1, 2, 3, 4, 7, 11], 'col2': [4, 5, 6, 9, 5, 0], 'col3': [7, 5, 8, 12, 1,11]}
df = pd.DataFrame(data=d)
print("Original DataFrame")
print(df)
print("\nAfter removing last 3 rows of the said DataFrame:")
df1 = df.iloc[:3]
print(df1)
```

Ans 117.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\coalpublic2013.xlsx')
print(df.head)
```

Ans 118.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\coalpublic2013.xlsx')
df[df["Mine_Name"].map(lambda x: x.startswith('P'))].head()
```

Ans 119.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\employee.xlsx')
df[df['hire_date'] >='20070101']
```

Ans 120.

```
import pandas as pd
import numpy as np
df = pd.read_excel('E:\employee.xlsx')
df2 = df.set_index(['hire_date'])
result = df2["2005"]
result
```

Ans 121.

```
import pandas as pd
import numpy as np
df1 = pd.read_excel('E:\employee.xlsx',sheet_name=0)
df2 = pd.read_excel('E:\employee.xlsx',sheet_name=1)
df3 = pd.read_excel('E:\employee.xlsx',sheet_name=2)
df = pd.concat([df1, df2, df3])
print(df)
```

Ans 122.

```
import pandas as pd
import numpy as np
df1 = pd.read_excel('E:\employee.xlsx',sheet_name=0)
df2 = pd.read_excel('E:\employee.xlsx',sheet_name=1)
df3 = pd.read_excel('E:\employee.xlsx',sheet_name=2)
df = pd.concat([df1, df2, df3])
df.to_excel('e:\output.xlsx', index=False)
```

Ans 123.

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
result = pd.pivot_table(df, index = ["sex","age"], aggfunc=np.sum)
print(result)
```

Ans 124.

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
result=df.groupby('sex')[['survived']].mean()
print(result)
```

Ans 125.

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
result = pd.cut(df['age'], [0, 10, 30, 60, 80])
print(result)
```

Ans 126.

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
age = pd.cut(df['age'], [0, 20, 55])
result = df.pivot_table('survived', index=['sex', age], columns='class')
print(result)
```

Ans 127

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
result = df.pivot_table(index=['sex'], columns=['pclass'], aggfunc='count')
print(result)
```

Ans 128.

```
import pandas as pd
import numpy as np
df = pd.read_csv('titanic.csv')
result = df.pivot_table( 'survived' , [ 'sex' , 'alone' ] , 'class' )
print(result)
```

Ans 129.

```
import pandas as pd
import numpy as np
```

```
df = pd.read_csv('titanic.csv')
result = df.pivot_table('survived', ['sex' , 'alone' ], [ 'embark_town', 'class' ])
print(result)
```

Ans 130

```
import pandas as pd
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
print("Original Dataframe:")
print(df.head())
print("\nCurrent date of Ufo dataset:")
print(df.Date_time.max())
print("\nOldest date of Ufo dataset:")
print(df.Date_time.min())
print("\nNumber of days between Current date and oldest date of Ufo dataset:")
print((df.Date_time.max() - df.Date_time.min()).days)
```

Ans 131.

```
import pandas as pd
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
print("Original Dataframe:")
print(df.head())
print("\nSighting days of the unidentified flying object (ufo) between 1949-10-10
and 1960-10-10:")
selected_period = df[(df['Date_time'] >= '1950-01-01 00:00:00') & (df['Date_time']
<= '1960-12-31 23:59:59')]
print(selected_period)
```

Ans 132

```
import pandas as pd
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
print("Original Dataframe:")
print(df.head())
print("\nYear:")
print(df.Date_time.dt.year.head())
print("\nMonth:")
print(df.Date_time.dt.month.head())
print("\nDay:")
print(df.Date_time.dt.day.head())
print("\nHour:")
print(df.Date_time.dt.hour.head())
print("\nMinute:")
print(df.Date_time.dt.minute.head())
print("\nSecond:")
print(df.Date_time.dt.second.head())
print("\nWeekday:")
print(df.Date_time.dt.weekday_name.head())
```

Ans 133.

```
import pandas as pd

df = pd.read_csv(r'ufo.csv')

df['Date_time'] = df['Date_time'].astype('datetime64[ns]')

print("Original Dataframe:")

print(df.head())

df['Year'] = df['Date_time'].apply(lambda x: "%d" % (x.year))

result = df.groupby(['Year', 'country']).size()

print("\nCountry-year wise frequency of reporting dates of UFO:")

print(result)
```

Ans 134.

```
import pandas as pd
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
df['date_documented'] = df['date_documented'].astype('datetime64[ns]')
print("Original Dataframe:")
print(df.head())
print("\nDifference (in days) between documented date and reporting date of
UFO:")
df['Difference'] = (df['date_documented'] - df['Date_time']).dt.days
print(df)
```

Ans 135.

```
import pandas as pd
```

```
dtr = pd.date_range('2018-01-01', periods=12, freq='H')
print("Hourly frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='min')
print("\nMinutely frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='S')
print("\nSecondly frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='2H')
print("nMultiple Hourly frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='5min')
print("\nMultiple Minutely frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='BQ')
print("\nMultiple Secondly frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='w')
print("\nWeekly frequency:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='2h20min')
print("\nCombine together day and intraday offsets-1:")
print(dtr)
dtr = pd.date_range('2018-01-01', periods=12, freq='1D10U')
print("\nCombine together day and intraday offsets-2:")
print(dtr)
```

Ans 136.

```
import pandas as pd
dtt = pd.date_range('2018-01-01', periods=3, freq='H')
dtt = dtt.tz_localize('UTC')
print(dtt)
print("\nFrom UTC to America/Los_Angeles:")
dtt = dtt.tz_convert('America/Los_Angeles')
print(dtt)
```

Ans 137

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
df["ufo_yr"] = df.Date_time.dt.year
years_data = df.ufo_yr.value_counts()
years_index = years_data.index  # x ticks
years_values = years_data.get_values()
plt.figure(figsize=(15,8))
plt.xticks(rotation = 60)
plt.title('UFO Sightings by Year')
plt.xlabel("Year")
plt.ylabel("Number of reports")
years_plot = sns.barplot(x=years_index[:60],y=years_values[:60], palette = "Reds")
```

Ans 138.

```
import pandas as pd
#Source: https://bit.ly/1l9yjm9
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
most_sightings_years = df['Date_time'].dt.year.value_counts().head(10)
def is_top_years(year):
  if year in most_sightings_years.index:
      return year
month_vs_year =
df.pivot_table(columns=df['Date_time'].dt.month,index=df['Date_time'].dt.year.app
ly(is_top_years),aggfunc='count',values='city')
month_vs_year.index = month_vs_year.index.astype(int)
month_vs_year.columns = month_vs_year.columns.astype(int)
print("\nComparison of the top 10 years in which the UFO was sighted vs each
month:")
print(month_vs_year.head(10))
```

Ans 139.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#Source: https://bit.ly/1l9yjm9
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
most_sightings_years = df['Date_time'].dt.year.value_counts().head(10)
def is_top_years(year):
  if year in most_sightings_years.index:
    return year
month_vs_year =
df.pivot_table(columns=df['Date_time'].dt.month,index=df['Date_time'].dt.year.app
ly(is_top_years),aggfunc='count',values='city')
month_vs_year.columns = month_vs_year.columns.astype(int)
print("\nHeatmap for comparison of the top 10 years in which the UFO was sighted
vs each month:")
plt.figure(figsize=(10,8))
ax = sns.heatmap(month_vs_year, vmin=0, vmax=4)
ax.set_xlabel('Month').set_size(20)
ax.set_ylabel('Year').set_size(20)
```

Ans 140

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.cm as cm
#Source: https://bit.ly/2XDY2XN
df = pd.read_csv(r'ufo.csv')
df['Date_time'] = df['Date_time'].astype('datetime64[ns]')
most_sightings_years = df['Date_time'].dt.year.value_counts().head(10)
def is_top_years(year):
  if year in most_sightings_years.index:
    return year
month_vs_year =
df.pivot_table(columns=df['Date_time'].dt.month,index=df['Date_time'].dt.year.app
ly(is_top_years),aggfunc='count',values='city')
month_vs_year.index = month_vs_year.index.astype(int)
month_vs_year.columns = month_vs_year.columns.astype(int)
print("\nComparison of the top 10 years in which the UFO was sighted vs each
month:")
def pie_heatmap(table, cmap='coolwarm_r', vmin=None,
vmax=None,inner_r=0.25, pie_args={}):
  n, m = table.shape
  vmin= table.min().min() if vmin is None else vmin
  vmax= table.max().max() if vmax is None else vmax

  centre_circle =
plt.Circle((0,0),inner_r,edgecolor='black',facecolor='white',fill=True,linewidth=0.2
5)
  plt.gcf().gca().add_artist(centre_circle)
  norm = mpl.colors.Normalize(vmin=vmin, vmax=vmax)
  cmapper = cm.ScalarMappable(norm=norm, cmap=cmap)

  for i, (row_name, row) in enumerate(table.iterrows()):
      labels = None if i > 0 else table.columns
      wedges = plt.pie([1] * m,radius=inner_r+float(n-i)/n,
colors=[cmapper.to_rgba(x) for x in row.values],
          labels=labels, startangle=90, counterclock=False, wedgeprops={'linewidth':-
1}, **pie_args)
      plt.setp(wedges[0], edgecolor='grey',linewidth=1.5)
      wedges = plt.pie([1], radius=inner_r+float(n-i-1)/n, colors=['w'],
labels=[row_name], startangle=-90, wedgeprops={'linewidth':0})
      plt.setp(wedges[0], edgecolor='grey',linewidth=1.5)
plt.figure(figsize=(8,8))
plt.title("Timewheel of Hour Vs Year",y=1.08,fontsize=30)
pie_heatmap(month_vs_year, vmin=-20,vmax=80,inner_r=0.2)
```

Ans 141.

```
import matplotlib.pyplot as plt
# x axis values
x = [1,2,3]
# y axis values
y = [2,4,1]
# Plot lines and/or markers to the Axes.
plt.plot(x, y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Sample graph!')
# Display a figure.
plt.show()
```

Ans 142.

```
import matplotlib.pyplot as plt

import pandas as pd

df = pd.read_csv('fdata.csv', sep=',', parse_dates=True, index_col=0)

df.plot()

plt.show()
```

Ans 143.

```
import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# plotting the line 1 points
plt.plot(x1, y1, label = "line 1")
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title of the current axes.
plt.title('Two or more lines on same plot with suitable legends ')
# show a legend on the plot
plt.legend()
# Display a figure.
plt.show()
```

Ans 144.

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, popularity, color='blue')
plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017
compared to a year ago")
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

Ans 145.

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JS', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.barh(x_pos, popularity, color='green')
plt.xlabel("Popularity")
plt.ylabel("Languages")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017
compared to a year ago")
plt.yticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

Ans 146.

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, popularity, color=(0.4, 0.6, 0.8, 1.0))
plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017
compared to a year ago")
# Rotation of the bars names
plt.xticks(x_pos, x, rotation=90)
# Custom the subplot layout
plt.subplots_adjust(bottom=0.4, top=.8)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

Ans 147.

```
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np

a=np.array([[4,8,5,7,6],[2,3,4,2,6],[4,7,4,7,8],[2,6,4,8,6],[2,4,3,3,2]])
df=DataFrame(a, columns=['a','b','c','d','e'], index=[2,4,6,8,10])

df.plot(kind='bar')
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

plt.show()
```

Ans 148.

```
import matplotlib.pyplot as plt
import pandas as pd
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```

Ans 149.

```
import matplotlib.pyplot as plt

import numpy as np

weight1=[67,57.2,59.6,59.64,55.8,61.2,60.45,61,56.23,56]

height1=[101.7,197.6,98.3,125.1,113.7,157.7,136,148.9,125.3,114.9]

weight2=[61.9,64,62.1,64.2,62.3,65.4,62.4,61.4,62.5,63.6]

height2=[152.8,155.3,135.1,125.2,151.3,135,182.2,195.9,165.1,125.1]
```

```
weight3=[68.2,67.2,68.4,68.7,71,71.3,70.8,70,71.1,71.7]

height3=[165.8,170.9,192.8,135.4,161.4,136.1,167.1,235.1,181.1,177.3]

weight=np.concatenate((weight1,weight2,weight3))

height=np.concatenate((height1,height2,height3))

plt.scatter(weight, height, marker='*', color=['red','green','blue'])

plt.xlabel('weight', fontsize=16)

plt.ylabel('height', fontsize=16)

plt.title('Group wise Weight vs Height scatter plot',fontsize=20)

plt.show()
```

Ans 150.

```
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('data.csv')
year = data['year']
sea_levels = data['CSIRO_sea_level']
plt.scatter(year, sea_levels, edgecolors='g')
plt.xlabel('Year')
plt.ylabel('Sea Level (inches)')
plt.title('Rise in Sealevel')
plt.show()
```