Week-6, Activity

Note

The `Scores Dataset` from the CT course is represented as a list of dictionaries and is provided as a lesson on the portal. Copy that list and paste it at the beginning of your code in all questions that involve the dataset.

Problem-1

Execute the following code. Why do you think this happens?

```python
def remove():
    L.pop()

L = list(range(5))
print('before:', L)
remove()
print('after:', L)
```

Answer

Two forces work together.

- `L` has global scope.
- `L` is a mutable object.

Because of these two reasons, we can modify the list within a function without explicitly passing it as an argument into it. Note that we are not merely referencing the list `L` here but also altering its contents. The important point is, we are not changing the memory location that the name `L` refers to.

Problem-2

Execute the following code. Why do you think this happens?

```
1  P = list(range(10))
2  Q = P
3  Q[0] = 100
4  print(P == Q)
5  print(P is Q)
```

Answer

The assignment statement in line-2 doesn't create a new list object. Instead, it merely creates another name to refer to the list `P`. More precisely, `P` and `Q` are two different names for the same list object. `Q` is an alias for `P`, that is, it is another name for the list that `P` points to. Think about an alias as a nickname.

Due to this reason, modifying the list `Q` is the same as modifying `P`. But one has to be careful here. This aliasing works only so long as `P` and `Q` are not reassigned values in a new assignment statement. For example, the relationship breaks down at line-6 in the following code:

```
1  P = list(range(10))
2  Q = P
3  Q[0] = 100
4  print(P == Q)
5  print(P is Q)
6  Q = 1
7  print(P == Q)
8  print(P is Q)
```

Problem-3

Execute the following code. Why do you think this happens?

```
1  A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
2  B = A.copy()
3  B[0][0] = 100
4  print(A == B)
5  print(A is B)
```

Answer

The whole idea of using the `copy` method is to create a new copy of a mutable object, so that modifying one doesn't modify the other. This is what we try to do in line-2. Surprisingly, even after creating a copy of `A` and storing it in `B`, modifying `B` affects the contents of `A`! This is because, `A.copy()` returns a new container to store the inner lists, while the inner lists continue to remain the same objects. This can be seen using the following statements:

```
for i in range(len(A)):
    print(A[i] is B[i])
```

In order to make a complete, penetrating copy, we need to take the help of a library named copy:

```
import copy
A = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
B = copy.deepcopy(A)
for i in range(len(A)):
    print(A[i] is B[i])
```

Now, we see that `A` and `B` and all the contents inside them are completely different objects. This kind of a copy is named `deepcopy` and we use the `deepcopy` function present in the `copy` library.

Problem-4

Execute the following code.

```python
D = dict()
for x in range(-10, 10):
    for y in range(-10, 10):
        if x ** 2 + y ** 2 - 25 < 0:
            D[(x, y)] = 'in'
        elif x ** 2 + y ** 2 - 25 == 0:
            D[(x, y)] = 'on'
        else:
            D[(x, y)] = 'out'
```

- What do you think is happening here?
- How many points are `in`, how many are `out` and how many are `on`?

Answer

- We are iterating through all the integer points in a 20×20 grid of points in the XY plane centered at the origin.
- For each point (x, y), we check if it is inside the circle, on it or outside it.
- The equation of the circle is given below:

$$x^2 + y^2 = 25$$

The conditions for a point to be inside, on and outside the circle is given below:

Condition	Position
$x^2 + y^2 < 25$	Inside
$x^2 + y^2 = 25$	On
$x^2 + y^2 > 25$	Outside

The code to find the number of points that are `in`, `out` and `on` the circle is given below:

```python
values = list(D.values())
in_count, out_count, on_count = values.count('in'), values.count('out'), values.count('on')

print(in_count, out_count, on_count)
```

Problem-5

`A`, `B`, `C` and `D` are four sets of numbers. Find the intersection and union of all four sets. A single line of code should do for each of the two cases. You can Google this information.

Answer

```
union = A | B | C | D
intersection = A & B & C & D
```

Problem-6

Construct the following sets in Python:

- `A` is the set of all positive integers less than or equal to 100 that are divisible by 3
- `B` is the set of all positive integers less than or equal to 100 that are divisible by 5.

Using Python's set notation, find the set of all integers that are:

- divisible by both 3 and 5
- divisible by 3 or 5
- divisible by 3 but not divisible by 5
- divisible by 5 but not divisible by 3

Note that each bullet corresponds to a separate set.

Answer

There are two ways of producing the sets `A` and `B`. It is good to know both ways of doing it.

```python
# method-1

A = set()
for i in range(3, 101, 3):
    A.add(i)

B = set()
for i in range(5, 101, 5):
    B.add(i)

# method-2
A = set(range(3, 101, 3))
B = set(range(5, 101, 5))
```

Likewise, there are two ways of computing the desired sets.

```python
# method-1
union = A | B
inter = A & B
diff1 = A - B
diff2 = B - A

# method-2
union = A.union(B)
inter = A.intersection(B)
diff1 = A.difference(B)
diff2 = B.difference(A)
```

Problem-7

Create a dictionary `D` with the following structure:

- `key`: numbers from 1 to 100, endpoints included
- `value`: set of factors of `key`

Using this information, find a pair of numbers in the range $[1, 100]$ that have the most number of factors in common. If there are multiple pairs, store all such pairs as a list of tuples.

Answer

```
### Compute D
D = dict()
for i in range(1, 101):
    D[i] = set()
    for j in range(1, i + 1):
        if i % j == 0:
            D[i].add(j)

### Compute set of common factors
### key is (i, j): a pair of integers
common = dict()
for i in range(1, 101):
    for j in range(i + 1, 101):
        common[(i, j)] = D[i] & D[j]

### Compute pair having maximum intersection
max_pairs, max_val = [ ], 0
for pair, com_pair in common.items():
    val = len(com_pair)
    if val > max_val:
        max_val = val
        max_pairs = [pair]
    if val == max_val and pair not in max_pairs:
        max_pairs.append(pair)

print(max_pairs)
```

Problem-8

Find an approximate solution to the following equation:

$$x^3 - 3x^2 + 2x - 1 = 0$$

Use Desmos to get an understanding of the initial value. This is not a mathematics questions, but a computational one. Think about how lists can be used to solve this problem. Once this is done, find the approximate value of x at which this curve attains a local maximum.

> Local maximum is a small bump in the curve that resembles a camel's hump.

Answer

When we plot the graph, we see that the solution lies somewhere in the range $(2, 3)$. So, the basic idea is to divide this unit line segment into a collection of points and then compute the function at each of these points.

```python
def f(x):
    return x ** 3 - 3 * x ** 2 + 2 * x - 1

### Create a linear grid
### We will systematically search through this grid
### limits is a tuple of start and end value
### step is the step size; how fine should the grid be
def grid(limits, step):
    points, p = [], limits[0]
    while p <= limits[1]:
        points.append(p)
        p += step
    # grids is a list of points
    return points

points = grid((2, 3), 0.1)
### Go through the grid
### Identify when f(x) turns from positive to negative
for x in points:
    print(f'{x:.2f} \t {f(x):.2f}')
```

Problem-9

Let `L` be a list of words. You are expected to create different kinds of dictionaries. In each case, think about the right choice of keys and their corresponding values.

- Create a dictionary that has information on the collection of words that have a specific letter count.
- Create a dictionary that has information the frequency of occurrence of words in the list `L`.
- Create a dictionary that contains information about the list of words that begin with a specific letter. Try to mimic the "English language dictionary" by sorting every list of words that begins with a given letter.

Answer

- letter count - set of words

```
# We will call the list as words
count = dict()
for word in words:
    # get the length of the word
    wlen = len(word)
    # if this length is not present in count
    # then create a key; value will be a set
    # set will store all words with this length
    if wlen not in count:
        count[wlen] = set()
    # we know for sure that wlen is a key
    # add this word to the set count[wlen]
    count[wlen].add(word)
```

- word - frequency of occurrence

```
freq = dict()
# prof already covered this in the lectures
for word in words:
    if word not in freq:
        freq[word] = 0
    freq[word] += 1
```

- character - list of words that begin with this character

```
eng_dict = dict()
for word in words:
    # first character in the word
    c = word[0]
    # if it is not yet a key
    # then add it as a key
    if c not in eng_dict:
        eng_dict[c] = set()
    # now that the key is present
    # add it to the set of words
    eng_dict[c].add(word)
# sort all words in alphabetical order
# sorted(set) will return a list of sorted items
```

```
for c in eng_dict:
    eng_dict[c] = sorted(eng_dict[c])
```

Problem-10

Extract the `Name` and `DateOfBirth` of all students from the `Scores Dataset` and store them as a list of tuples. Each tuple should be of the form: `(Name, DateOfBirth)`. For example, a truncated list of size 2 would look like this:

```
[ ('Bhuvanesh', '7 Nov'), ('Harish', '3 Jun') ]
```

Answer

```python
details = [ ]
for student in scores:
    details.append((student['Name'], student['DateOfBirth']))
```

Problem-11

Extract the `Physics` marks of all students from the `Scores Dataset` and store them in a list. Now, transfer the contents of this list into a set.

- Do you lose any information in this process?
- When would this operation be useful? Does any application spring to your mind?

Answer

```python
phy_list = [ ]
for student in scores:
    phy_list.append(student['Physics'])

phy_set = set(phy_list)
```

- We do lose information when converting a list to a set.
- This operation (conversion from list to set) would be useful when we are interested in finding out the number of unique occurrences of an item in a collection. On the other hand, if duplicates are important then this operation is a dangerous thing to do.

Problem-12

Consider the following graph generated from the `Scores Dataset`:

- Each student is represented by a node in the graph.
- There is an edge between two students i and j in the graph if they are from the same `CityTown`.

Construct the adjacency matrix corresponding to this graph. Solve the problem with these two approaches:

- nested lists
- nested dictionaries

Answer

The solution for nested lists is given below:

```python
def zero_matrix(dim):
    A = [ ]
    for i in range(dim):
        A.append([ ])
        for j in range(dim):
            A[-1].append(0)
    return A

def populate(adj_mat, scores):
    for si in range(len(scores)):
        for sj in range(len(scores)):
            # checking for edge condition
            if si != sj and scores[si]['CityTown'] == scores[sj]['CityTown']:
                adj_mat[si][sj] = 1
    return adj_mat

adj_mat = zero_matrix(len(scores))
adj_mat = populate(adj_mat, scores)
```

Problem-13

Extract the `Name` and `Mathematics` marks of all students from the `Scores Dataset` and store them as a list of tuples. Each tuple should be of the form: `(Name, Mathematics)`. Sort this list in ascending order of marks. If there are two students who have scored the same marks, then sort based on the `Name` (alphabetical order). Note that the final list should also be a list of tuples.

Answer

```python
data = [ ]

# Insert x into a list of tuples
# x itself is a tuple
# x[0] is name, x[1] is marks
def insert(L, x):
    out_L = [ ]
    inserted = False
    for elem in L:
        # elem[0] is name, elem[1] is marks
        # check if x has already been inserted
        if (not inserted):
            # first compare based on marks
            # if marks are equal, compare based on names
            if ((elem[1] > x[1]) or
                (elem[1] == x[1] and elem[0] > x[0])):
                out_L.append(x) # element inserted
                inserted = True
        out_L.append(elem)
    # corner case for empty list or last element
    if not inserted:
        out_L.append(x)
    return out_L

# we are first sorting by second index (marks)
# then we are sorting by first index (name)
# note that while sorting by name, we don't
# disturb the sorting order by marks
def isort(L):
    out_L = [ ]
    for elem in L:
        out_L = insert(out_L, elem)
    return out_L

# create list of tuples and store it in data
for student in scores:
    data.append((student['Name'], student['Mathematics']))

# use insertion sort to sort the list of tuples
sorted_data = isort(data)
for name, math in sorted_data:
    print(name, math)
```

Problem-14

Convert the scores dataset into a dictionary with the following structure:

- `key`: `SeqNo` of a student
- `value`: dictionary containing all the details of the student with the above `SeqNo`

Add a new field for `Biology` marks for each student. You can use the `random` library to randomly assign marks to students in this subject.

Answer

```python
from random import randint

data = dict()

for student in scores:
    seq_no = student['SeqNo']
    data[seq_no] = student
    data[seq_no]['Biology'] = randint(40, 100)
```

Problem-15

Consider a three-dimensional, graphical representation of the students from the `Scores Dataset`.

Axis	Entity
X	Mathematics marks
Y	Physics marks
Z	Chemistry marks

Using this representation, each student can be identified by a point (x, y, z) in space. The distance between any two students S_1 and S_2 is measured using the Manhattan formula:

$$D(S_1, S_2) = |x_1 - x_2| + |y_1 - y_2| + |z_1 - z_2|$$

where, (x_1, y_1, z_1) and (x_2, y_2, z_2) represent the coordinates of the two students respectively.

Task-1

Generate a list of dictionaries, where each dictionary has the details of a pair of students. Specifically, each dictionary should have the following information:

```
S1: Name
S2: Name
Distance: D(S1, S2)
```

Task-2

Sort this list based on the distance field. That is, the output should be a list of dictionaries, but sorted in ascending order of distance.

Task-3

Use this sorted list to find the pair of students who are:

- closest to each other
- farthest from each other

Answer

Task-1

```python
# distance function needed here
# we will be reusing it heavily
def distance(s1, s2):
    return abs(s1[0] - s2[0]) + abs(s1[1] - s2[1]) + abs(s1[2] - s2[2])

data = [ ]
size = len(scores) # num of students
for si in range(size):
    for sj in range(si + 1, size):
        info = dict()   # each pair goes into info dict
        # (x1, y1, z1)
        s1 = (scores[si]['Mathematics'], scores[si]['Physics'], scores[si]['Chemistry'])
        # (x2, y2, z2)
        s2 = (scores[sj]['Mathematics'], scores[sj]['Physics'], scores[sj]['Chemistry'])
        # distance between s1 and s2
        dij = distance(s1, s2)
        info['S1'] = scores[si]['Name']
        info['S2'] = scores[sj]['Name']
        info['distance'] = dij
        data.append(info)
```

Task-2

```
# sorting  a list of dicts
# based on distance key in each dict
# x is a dict
def insert(L, x):
    out_L = [ ]
    inserted = False
    for elem in L:
        # elem is a dict
        if (not inserted) and elem['distance'] > x['distance']:
            out_L.append(x)
            inserted = True
        out_L.append(elem)
    if not inserted:
        out_L.append(x)
    return out_L

def isort(L):
    out_L = [ ]
    for elem in L:
        out_L = insert(out_L, elem)
    return out_L

sorted_data = isort(data)
```

Task-3

```
sorted_data[0]  # closest
sorted_data[-1] # farthest
```