Q1. Get your basics right - Implement selection sort algorithm in python. The function accepts a

list in the input and returns a sorted list. E.g. Input f1([5,416,54,21,6135,15,741]) should Return [5, 15, 21, 54, 416, 741, 6135]

In [1]:

```
def selection_sort(lst):
    for i in range(len(lst)):
        min_index = i
        for j in range(i + 1, len(lst)):
            if lst[j] < lst[min_index]:
                min_index = j
        lst[i], lst[min_index] = lst[min_index], lst[i]
    return lst
```

In [2]:

```
lst = [5, 416, 54, 21, 6135, 15, 741]
sorted_lst = selection_sort(lst)
print(sorted_lst)
```

```
[5, 15, 21, 54, 416, 741, 6135]
```

Q2. Dictionary, what?

Write a program that returns the file type from a file name. The type of the file is determined from the extension. Initially, a list of values of the form "extension,type"(e.g. xls,spreadsheet; png,image) will be input. The program takes input in the following form:

1. Input extension and type values in the form of a string having the following format: a. "extension1,type1;extension2,type2;extension3,type3" b. E.g. If we needed to input xls → spreadsheet, xlsx → spreadsheet, jpg → image our string would be something like: "xls,spreadsheet;xlsx,spreadsheet;jpg,image"
2. Input a list of filename.extension. E.g. an input list could be something like ["abc.html", "xyz.xls", "text.csv", "123"] The program should return a dict of filename: type pairs E.g. f("xls,spreadsheet;xlsx,spreadsheet;jpg,image", ["abc.jpg", "xyz.xls", "text.csv", "123"]) should return { "abc.jpg": "image", "xyz.xls": "spreadsheet", "Text.csv": "unknown", "123": "unknown" }

To solve this problem, we can follow these steps:

Parse the input string containing the extension and type values into a dictionary. Iterate over the list of filenames and extract the extension for each filename. Use the extracted extension to look up the corresponding file type in the dictionary. Store the filename and its associated file type in a dictionary. Return the final dictionary of filename: type pairs. Here's the implementation of the program in Python:

In [3]:

```python
def get_file_types(extension_types, filenames):
    # Parse the extension and type values into a dictionary
    ext_type_dict = {}
    pairs = extension_types.split(';')
    for pair in pairs:
        extension, file_type = pair.split(',')
        ext_type_dict[extension] = file_type

    # Iterate over the filenames and determine the file type
    result = {}
    for filename in filenames:
        # Extract the extension from the filename
        parts = filename.split('.')
        if len(parts) > 1:
            extension = parts[-1]
        else:
            extension = ''

        # Look up the file type in the dictionary
        file_type = ext_type_dict.get(extension, 'unknown')
        result[filename] = file_type

    return result
```

In [4]:

```python
extension_types = "xls,spreadsheet;xlsx,spreadsheet;jpg,image"
filenames = ["abc.jpg", "xyz.xls", "text.csv", "123"]
result = get_file_types(extension_types, filenames)
print(result)
```

```
{'abc.jpg': 'image', 'xyz.xls': 'spreadsheet', 'text.csv': 'unknown', '12
3': 'unknown'}
```

3. Column Sorting, yay!

Given a list of dicts, write a program to sort the list according to a key given in input.

E.g. Input f([{"fruit": "orange", "color": "orange"}, {"fruit": "apple", "color": "red"}, {"fruit": "banana", "color": "yellow"}, {"fruit": "blueberry", "color": "blue"}], "fruit")

Should Output

[{"fruit": "apple", "color": "red"}, {"fruit": "banana", "color": "yellow"}, {"fruit": "blueberry", "color": "blue"}, {"fruit": "orange", "color": "orange"}] AND

Input f([{"fruit": "orange", "color": "orange"}, {"fruit": "apple", "color": "red"}, {"fruit": "banana", "color": "yellow"}, {"fruit": "blueberry", "color": "blue"}], "color")

Should Output

[{"fruit": "blueberry", "color": "blue"}, {"fruit": "orange", "color": "orange"}, {"fruit": "apple", "color": "red"}, {"fruit": "banana", "color": "yellow"}]

To sort a list of dictionaries based on a specified key, we can use the sorted function with a custom key function. The key function will extract the value of the specified key from each dictionary for comparison during sorting.

Here's the implementation of the program in Python:

In [5]:

```
def sort_list_of_dicts(lst, key):
    sorted_lst = sorted(lst, key=lambda x: x[key])
    return sorted_lst
```

In [6]:

```
lst = [
    {"fruit": "orange", "color": "orange"},
    {"fruit": "apple", "color": "red"},
    {"fruit": "banana", "color": "yellow"},
    {"fruit": "blueberry", "color": "blue"}
]
key = "fruit"
sorted_lst = sort_list_of_dicts(lst, key)
print(sorted_lst)
```

```
[{'fruit': 'apple', 'color': 'red'}, {'fruit': 'banana', 'color': 'yello
w'}, {'fruit': 'blueberry', 'color': 'blue'}, {'fruit': 'orange', 'color':
'orange'}]
```

In [7]:

```
lst = [
    {"fruit": "orange", "color": "orange"},
    {"fruit": "apple", "color": "red"},
    {"fruit": "banana", "color": "yellow"},
    {"fruit": "blueberry", "color": "blue"}
]
key = "color"
sorted_lst = sort_list_of_dicts(lst, key)
print(sorted_lst)
```

```
[{'fruit': 'blueberry', 'color': 'blue'}, {'fruit': 'orange', 'color': 'or
ange'}, {'fruit': 'apple', 'color': 'red'}, {'fruit': 'banana', 'color':
'yellow'}]
```

4.The power of one line -

Given a dictionary, switch position of key and values in the dict, i.e., value becomes the key and key becomes value. The function's body shouldn't have more than one statement.

f({ "key1": "value1", "key2": "value2", "key3": "value3", "key4": "value4", "key5": "value5" }) should return { "value1": "key1", "value2": "key2", "value3": "key3", "value4": "key4", "value5": "key5" }

In [8]:

```
def switch_dict(dict_input):
    return {value: key for key, value in dict_input.items()}
```

In [9]:

```
dict_input = {
    "key1": "value1",
    "key2": "value2",
    "key3": "value3",
    "key4": "value4",
    "key5": "value5"
}
result = switch_dict(dict_input)
print(result)
```

```
{'value1': 'key1', 'value2': 'key2', 'value3': 'key3', 'value4': 'key4',
'value5': 'key5'}
```

Q5. Common, Not Common

Given 2 lists in input. Write a program to return the elements, which are common to both lists(set intersection) and those which are not common(set symmetric difference) between the lists. Input:

Mainstream = ["One Punch Man","Attack On Titan","One Piece","Sword Art Online","Bleach","Dragon Ball Z","One Piece"] must_watch = ["Full Metal Alchemist","Code Geass","Death Note","Stein's Gate","The Devil is a Part Timer!","One Piece","Attack On Titan"]

f(mainstream, must_watch) should return: ["One Piece", "Attack On Titan"], ["Dragon Ball Z", "Death Note", "One Punch Man", "Stein's Gate", "The Devil is a Part Timer!", "Sword Art Online","Full Metal Alchemist","'Bleach", "Code Geass"]

In [10]:

```
def find_common_and_uncommon(list1, list2):
    set1 = set(list1)
    set2 = set(list2)

    common = list(set1 & set2)
    uncommon = list(set1 ^ set2)

    return common, uncommon
```

In [11]:

```
Mainstream = ["One Punch Man", "Attack On Titan", "One Piece", "Sword Art Online", "Blea
must_watch = ["Full Metal Alchemist", "Code Geass", "Death Note", "Stein's Gate", "The D

common, uncommon = find_common_and_uncommon(Mainstream, must_watch)
print(common)
print(uncommon)
```

```
['Attack On Titan', 'One Piece']
['One Punch Man', 'Full Metal Alchemist', 'The Devil is a Part Timer!', 'D
eath Note', "Stein's Gate", 'Sword Art Online', 'Dragon Ball Z', 'Code Gea
ss', 'Bleach']
```

Q6. Every other sub-list

Given a list and 2 indices as input, return the sub-list enclosed within these 2 indices. It should contain every second element.

E.g. Input f([2,3,5,7,11,13,17,19,23,29,31,37,41], 2, 9) Return [5, 11, 17, 23]

In [12]:

```
def extract_sublist(lst, start_index, end_index):
    sub_list = lst[start_index:end_index:2]
    return sub_list
```

In [13]:

```
lst = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]
start_index = 2
end_index = 9

result = extract_sublist(lst, start_index, end_index)
print(result)
```

```
[5, 11, 17, 23]
```

Q7. Calculate the factorial of a number using lambda function.

In [14]:

```
factorial = lambda n: 1 if n == 0 else n * factorial(n - 1)
```

In [16]:

```
number = 5
result = factorial(number)
print(result)
```

```
120
```

Q8. Some neat tricks up her sleeve:

Looking at the below code, write down the final values of A0, A1, ...An

A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5))) A1 = range(10) A2 = sorted([i for i in A1 if i in A0]) A3 = sorted([A0[s] for s in A0]) A4 = [i for i in A1 if i in A3] A5 = {i:i*i for i in A1} A6 = [[i,i*i] for i in A1] A7 = reduce(lambda x,y: x+y, [10,23, -45, 33]) A8 = map(lambda x: x*2, [1,2,3,4]) A9 = filter(lambda x: len(x) >3, [“I” , “want”, “to”, “learn”, “python”])

In [32]:

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
```

In [33]:

```
A0
```

Out[33]:

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

In [42]:

```
import numpy as np
```

In [43]:

```
A = np.arange(10)
A
```

Out[43]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [49]:

```
A2 = sorted([i for i in A if i in A0.values()])
A2
```

Out[49]:

```
[1, 2, 3, 4, 5]
```

In [38]:

```
A3 = sorted([A0[s] for s in A0])
A3
```

Out[38]:

```
[1, 2, 3, 4, 5]
```

In [39]:

```
A4 = [i for i in A1 if i in A3]
A4
```

Out[39]:

```
[1, 2, 3, 4, 5]
```

In [40]:

```
A5 = {i: i*i for i in A1}
A5
```

Out[40]:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

In [41]:

```
A6 = [[i, i*i] for i in A1]
A6
```

Out[41]:

```
[[0, 0],
 [1, 1],
 [2, 4],
 [3, 9],
 [4, 16],
 [5, 25],
 [6, 36],
 [7, 49],
 [8, 64],
 [9, 81]]
```

In [54]:

```
import functools
A7 = functools.reduce(lambda x, y: x + y, [10, 23, -45, 33])
A7
```

Out[54]:

```
21
```

In [58]:

```
A8 = map(lambda x: x*2, [1, 2, 3, 4])
list(A8)
```

Out[58]:

```
[2, 4, 6, 8]
```

So, the final values are:

A0: {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5} A1: range(0, 10) A2: [1, 2, 3, 4, 5] A3: [1, 2, 3, 4, 5] A4: [1, 2, 3, 4, 5] A5: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81} A6: [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64], [9, 81]] A7: 21 A8: [2, 4, 6, 8] A9: ['want', 'learn', 'python']

Q9.

Write a func that takes 3 args: from_date - string representing a date in the form of 'yy-mm-dd' to_date - string representing a date in the form of 'yy-mm-dd' difference - int Returns True if from_date and to_date are less than difference days away from each other, else returns False.

In [59]:

```
from datetime import datetime, timedelta

def check_date_difference(from_date, to_date, difference):
    # Convert the date strings to datetime objects
    from_datetime = datetime.strptime(from_date, '%y-%m-%d')
    to_datetime = datetime.strptime(to_date, '%y-%m-%d')

    # Calculate the difference between the dates
    date_difference = to_datetime - from_datetime

    # Check if the difference is less than the specified number of days
    if date_difference < timedelta(days=difference):
        return True
    else:
        return False
```

In [60]:

```
from_date = '21-01-01'
to_date = '21-01-15'
difference = 20

result = check_date_difference(from_date, to_date, difference)
print(result)  # Output: True

from_date = '21-01-01'
to_date = '21-01-15'
difference = 10

result = check_date_difference(from_date, to_date, difference)
print(result)  # Output: False
```

```
True
False
```

In the above examples, the function checks if the difference between the from_date and to_date is less than the specified difference in days. The function returns True if the difference is less and False otherwise.

Q10. Of date and days

Write a func that takes 2 args: date - string representing a date in the form of 'yy-mm-dd' n - integer Returns the string representation of date n days before 'date' E.g. f('16-12-10', 11) should return '16-11-29'

In [61]:

```
from datetime import datetime, timedelta

def calculate_previous_date(date, n):
    # Convert the date string to a datetime object
    date_obj = datetime.strptime(date, '%y-%m-%d')

    # Calculate the previous date by subtracting n days
    previous_date = date_obj - timedelta(days=n)

    # Convert the datetime object back to a string representation
    previous_date_str = previous_date.strftime('%y-%m-%d')

    return previous_date_str
```

In [62]:

```
date = '16-12-10'
n = 11

result = calculate_previous_date(date, n)
print(result)  # Output: '16-11-29'
```

```
16-11-29
```

In the above example, the function calculates the date that is n days before the given date. The function returns the string representation of the calculated previous date.

Please note that the function assumes the input date format is always 'yy-mm-dd' and the output date format will also be 'yy-mm-dd'.

Q11. Something fishy there -

Find output of following: def f(x,l=[]): for i in range(x): l.append(i*i) print(l) f(2) f(3,[3,2,1]) f(3)

In [72]:

```
def fishy(x, l1=[]):
    for i in range(x):
        l1.append(i*i)
    print(l1)

fishy(2)
```

```
[0, 1]
```

In [73]:

```
fishy(3, [3, 2, 1])
```

```
[3, 2, 1, 0, 1, 4]
```

In [74]:

```
fishy(3)
```

```
[0, 1, 0, 1, 4]
```

In [ ]:

```

```