Softy EDUCATION

&

Taki Academy
www.takiacademy.com

python™

contact@softyeducation.com

Softy EDUCATION

Day 5

python™

Content :

1. Tuple
2. Set
3. List
4. Dictionary
5. Exercises

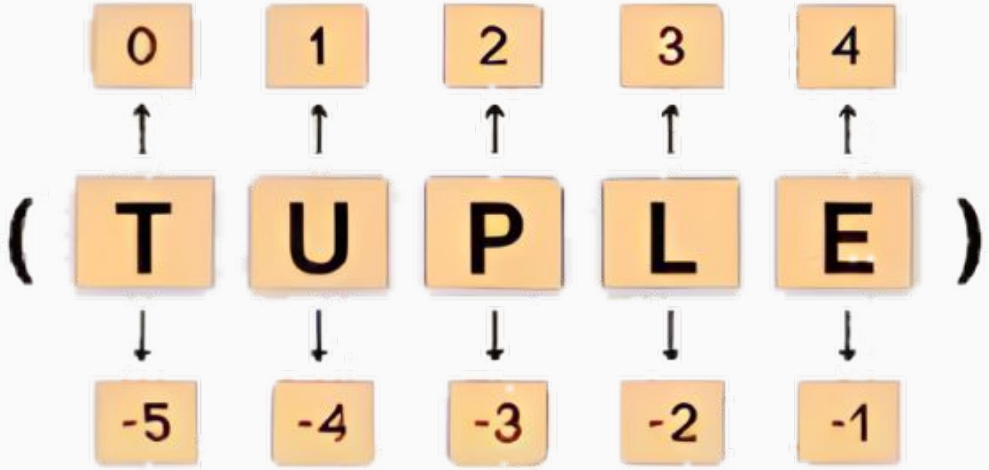
Softy EDUCATION

01 Tuples in Python

python™

01- Tuples in Python

Tuples are immutable ordered sequences of elements. So, their content cannot be changed after they are created.



01- Tuples in Python

You can create a tuple by placing a comma-separated sequence of values inside parentheses ().

app.py

```python
my_tuple = (1, 'apple', 3.14)
empty_tuple = ()
# note the comma after the element to indicate
#  it's a tuple and not an integer variable declaration
one_element_tuple = (1,)
```

01- Tuples in Python

Accessing Elements

You can access the elements of a tuple using indexing, just like with lists:

app.py

```
my_tuple = (1, 'apple', 3.14)
print(my_tuple[1])  # apple
```

01- Tuples in Python

Immutability

Once a tuple is created, you cannot modify its content. This means that the following code will raise an error:

app.py

```python
my_tuple = (1, 'apple', 3.14)
# TypeError: 'tuple' object does
#  not support item assignment
my_tuple[1] = 'banana'
```

01- Tuples in Python

In Python ,methods that add items or remove items are not available with tuple. Only the following two methods are available. Here,

my_tuple.count('p') - counts total number of 'p' in my_tuple

my_tuple.index('l') - returns the first occurrence of 'l' in my_tuple

app.py

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)

print(my_tuple.count('p'))  # prints 2
print(my_tuple.index('l'))  # prints 3
```

01- Tuples in Python

Iterating through a Tuple in Python :

We can use the **for loop** to iterate over the elements of a tuple. For example,

app.py

```
languages = ('Python', 'Swift', 'C++')

# iterating through the tuple
for language in languages:
    print(language)

#Python
#Swift
#C++
```

01- Tuples in Python

Check if an Item Exists in the Python Tuple :

We use the **in** keyword to check if an item exists in the tuple or not. For example,

app.py

```
languages = ('Python', 'Swift', 'C++')

print('C' in languages)     # False
print('Python' in languages)     # True
```

01- Tuples in Python

Advantages of Tuple over List in Python :

Since tuples are quite similar to lists, both of them are used in similar situations. However, there are certain advantages of implementing a tuple over a list:

- We generally use tuples for heterogeneous (different) data types and lists for homogeneous (similar) data types.
- Since tuples are immutable, iterating through a tuple is faster than with a list. So there is a slight performance boost.

Softy EDUCATION

02 Sets in Python

python™

02- Sets in python

A set is a collection of unique data. That is, elements of a set cannot be duplicate. For example,

Suppose we want to store information about **student IDs.** Since **student IDs** cannot be **duplicate**, we can use a **set.**

Set of Student ID

02- Sets in python

Create a Set in Python :

In Python, we create sets by placing all the elements inside curly braces **{}**, separated by comma.

A set can have elements with different types (integer, float, tuple, string etc.). But a set **cannot have mutable** elements like **lists, sets or dictionaries** .

02- Sets in python

Examples :

app.py

```python
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)

# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)

# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```

02- Sets in python

Output :

app.py

```
# Student ID: {112, 114, 115, 116, 118}
# Vowel Letters: {'u', 'a', 'e', 'i', 'o'}
# Set of mixed data types: {'Hello', 'Bye', 101, -2}
```

02- Sets in python

Create an Empty Set in Python :

Creating an empty set is a bit tricky. Empty curly braces {} will make an empty dictionary in Python.

To make a set without any elements, we use set() .

app.py

```
# create an empty set
empty_set = set()

# create an empty dictionary
empty_dictionary = { }

# check data type of empty_set
print('Data type of empty_set:', type(empty_set))
# Data type of empty_set: <class 'set'>

# check data type of dictionary_set
print('Data type of empty_dictionary', type(empty_dictionary))
# Data type of empty_dictionary <class 'dict'>
```

02- Sets in python

Duplicate Items in a Set :

Let's see what will happen if we try to include duplicate items in a set.

app.py

```
numbers = {2, 4, 6, 6, 2, 8}
print(numbers)    # {8, 2, 4, 6}
```

02- Sets in python

Add and Update Set Items in Python :

Sets are mutable. However, since they are unordered, indexing has no meaning.

We cannot access or change an element of a set using indexing or slicing.

02- Sets in python

1. Add Items to a Set in Python :

In Python, we use the add() method to add an item to a set. For example,

app.py

```
numbers = {21, 34, 54, 12}

print('Initial Set:',numbers)
# Initial Set: {34, 12, 21, 54}

# using add() method
numbers.add(32)

print('Updated Set:', numbers)
# Updated Set: {32, 34, 12, 21, 54}
```

02- Sets in python

2. Update Python Set :

The update() method is used to update the set with items other collection types (lists, tuples, sets, etc). For example,

app.py

```
companies = {'Lacoste', 'Ralph Lauren'}
tech_companies = ['apple', 'google', 'apple']

companies.update(tech_companies)

print(companies)

# Output: {'google', 'apple', 'Lacoste', 'Ralph Lauren'}
```

02- Sets in python

3. Remove an Element from a Set :

We use the discard() method to remove the specified element from a set. For example,

app.py

```
languages = {'Swift', 'Java', 'Python'}

print('Initial Set:',languages)
# Initial Set: {'Python', 'Swift', 'Java'}

# remove 'Java' from a set
removedValue = languages.discard('Java')

print('Set after remove():', languages)
# Set after remove(): {'Python', 'Swift'}
```

4. Built-in Functions with Set :

Built-in functions like **all(), any(), enumerate(), len(), max(), min(), sorted(), sum()** etc. are commonly used with sets to perform different tasks.

02- Sets in python

Function	Description
all()	Returns `True` if all elements of the set are true (or if the set is empty).
any()	Returns `True` if any element of the set is true. If the set is empty, returns `False`.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.

Softy EDUCATION

03 Lists in Python

python™

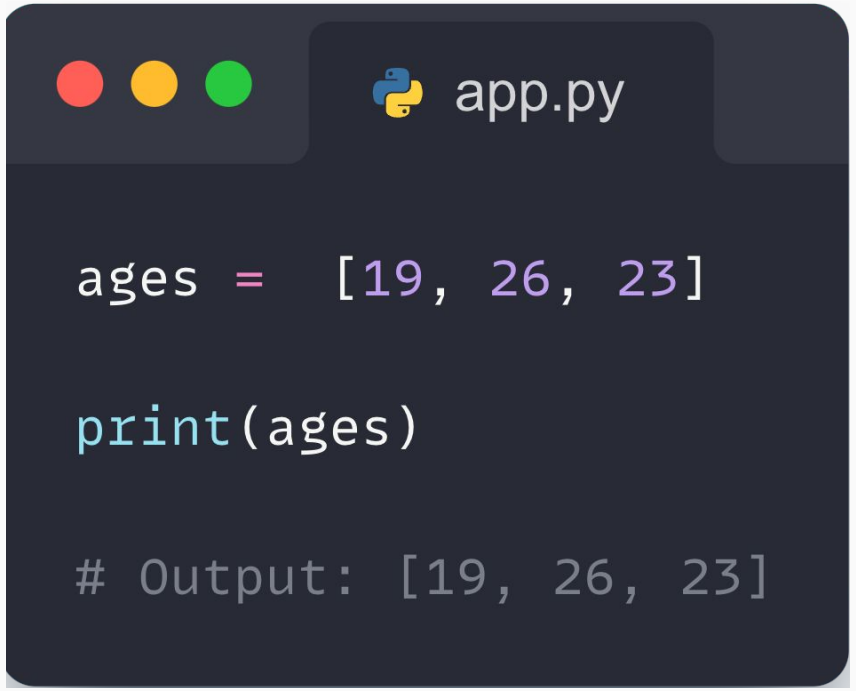
03- Lists in python

In Python, lists are used to store multiple data at once. Suppose we need to record the **ages** of **5 students**. Instead of creating **5 separate variables**, we can simply **create a list.**

03- Lists in python

Create a List :

We create a list by placing elements inside [], separated by commas. For example,



03- Lists in python

A list can :

- store elements of different types (integer, float, string, etc.)
- store duplicate elements

app.py

```
# list with elements of different data types
list1 = [1, "Hello", 3.4]

# list with duplicate elements
list1 = [1, "Hello", 3.4, "Hello", 1]

# empty list
list3 = []
```

03- Lists in python

Access List Elements :

In Python, lists are ordered and each item in a list is associated with a number. The number is known as a list index.

The index of the first element is 0, second element is 1 and so on.

For example,

app.py

```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[0])   # Python

# access item at index 2
print(languages[2])   # C++
```

03- Lists in python

In the above example, we have created a list named languages.

03- Lists in python

Negative Indexing in Python :

Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

app.py

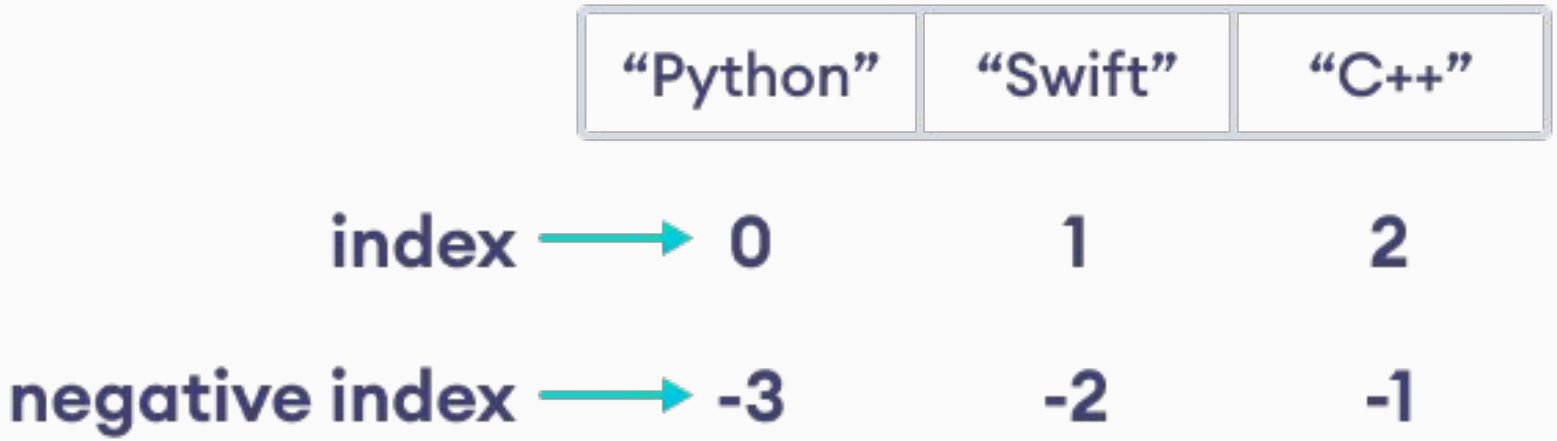
```
languages = ["Python", "Swift", "C++"]

# access item at index 0
print(languages[-1])   # C++

# access item at index 2
print(languages[-3])   # Python
```

03- Lists in python

Negative Indexing in Python :



03- Lists in python

Slicing of a List :

In Python, it is possible to access a portion of a list using the slicing operator **:**

For example,

app.py

```python
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# items from index 2 to index 4
print(my_list[2:5])
# ['o', 'g', 'r']

# items from index 5 to end
print(my_list[5:])
# ['a', 'm', 'i', 'z']

# items beginning to end
print(my_list[:])
# ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

Add Elements to a List :

Lists are mutable (changeable). Meaning we can add and remove elements from a list.

03- Lists in python

1. Using append() :

The append() method adds an item at the end of the list. For example,

app.py

```python
numbers = [21, 34, 54, 12]

print("Before Append:", numbers)
# Before Append: [21, 34, 54, 12]

# using append method
numbers.append(32)

print("After Append:", numbers)
# After Append: [21, 34, 54, 12, 32]
```

2. Using extend() :

We use the extend() method to add all the items of an iterable (list, tuple, string, dictionary, etc.) to the end of the list. For example,

app.py

```
numbers = [1, 3, 5]

even_numbers = [4, 6, 8]

# add elements of even_numbers to the numbers list
numbers.extend(even_numbers)

print("List after append:", numbers)
# List after append: [1, 3, 5, 4, 6, 8]
```

3. Using insert() :

We use the insert() method to add an element at the specified index.

app.py

```
numbers = [10, 30, 40]

# insert an element at index 1 (second position)
numbers.insert(1, 20)

print(numbers) # [10, 20, 30, 40]
```

03- Lists in python

Change List Items :

Python lists are mutable. Meaning lists are changeable. And we can change items of a list by assigning new values using the **=** operator.

For example,

app.py

```
languages = ['Python', 'Swift', 'C++']

# changing the third item to 'C'
languages[2] = 'C'

print(languages)  # ['Python', 'Swift', 'C']
```

03- Lists in python

Remove an Item From a List :

1. Using **del** Statement :

   In Python we can use the **del** statement to remove one or more items from a list. For example,

app.py

```python
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the second item
del languages[1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the last item
del languages[-1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']

# delete the first two items
del languages[0 : 2]  # ['C', 'Java', 'Rust']
print(languages)
```

03- Lists in python

2. Using **remove** Statement :

   We can also use the **remove()** method to delete a list item. For example,

app.py

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']

# remove 'Python' from the list
languages.remove('Python')

print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust', 'R']
```

03- Lists in python

List Methods :

Method	Description
append()	add an item to the end of the list
extend()	add all the items of an iterable to the end of the list
insert()	inserts an item at the specified index
remove()	removes item present at the given index
pop()	returns and removes item present at the given index
clear()	removes all items from the list
index()	returns the index of the first matched item
count()	returns the count of the specified item in the list
sort()	sort the list in ascending/descending order
reverse()	reverses the item of the list
copy()	returns the shallow copy of the list

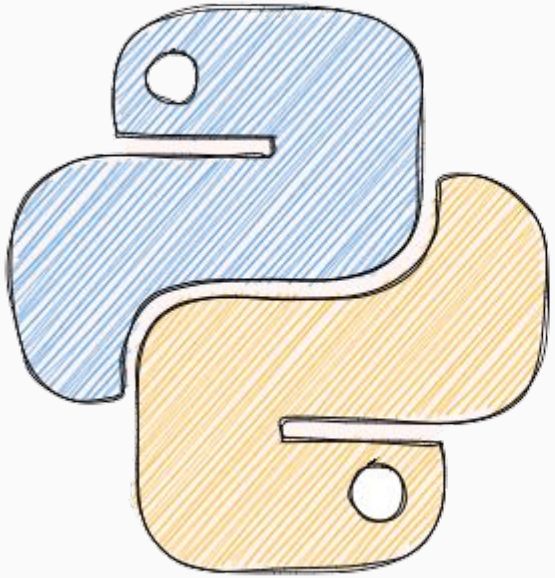
Softy EDUCATION

04 Dictionaries in Python

python™

04- Dictionaries in python

In Python, a dictionary is a collection that allows us to store data in key-value pairs.



04- Dictionaries in python

Create a Dictionary :

We create dictionaries by placing key:value pairs inside curly brackets **{}**, separated by commas. For example,

app.py

```
# creating a dictionary
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
# {'United States': 'Washington D.C.',
# 'Italy': 'Rome',
# 'England': 'London'}
```

04- Dictionaries in python

Create a Dictionary :

We create dictionaries by placing key:value pairs inside curly brackets **{}**, separated by commas. For example,

app.py

```
# creating a dictionary
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# printing the dictionary
print(country_capitals)
# {'United States': 'Washington D.C.',
# 'Italy': 'Rome',
# 'England': 'London'}
```

04- Dictionaries in python

Note :

Dictionary keys must be immutable, such as tuples, strings, integers, etc. We cannot use mutable (changeable) objects such as lists as keys.

app.py

```python
# Valid dictionary

my_dict = {
  1: "Hello",
  (1, 2): "Hello Hi",
  3: [1, 2, 3]
}

print(my_dict)

# Invalid dictionary
# Error: using a list as a key is not allowed

my_dict = {
  1: "Hello",
  [1, 2]: "Hello Hi",
}

print(my_dict)
```

04- Dictionaries in python

Python Dictionary Length :

We can get the size of a dictionary by using the **len()** function.

app.py

```
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

# get dictionary's length
print(len(country_capitals)) # 3
```

04- Dictionaries in python

Access Dictionary Items :

We can access the value of a dictionary item by placing the key inside square brackets.

app.py

```
country_capitals = {
  "United States": "Washington D.C.",
  "Italy": "Rome",
  "England": "London"
}

print(country_capitals["United States"])  # Washington D.C.

print(country_capitals["England"]) # London
```

04- Dictionaries in python

Change Dictionary Items :

Python dictionaries are mutable (changeable). We can change the value of a dictionary element by referring to its key. For example,

app.py

```
country_capitals = {
  "United States": "New York",
  "Italy": "Naples",
  "England": "London"
}

# change the value of "Italy" key to "Rome"
country_capitals["Italy"] = "Rome"

print(country_capitals)
# {'United States': 'Washington D.C.',
# 'Italy': 'Rome',
# 'England': 'London'}
```

04- Dictionaries in python

Add Items to a Dictionary :

We can add an item to the dictionary by assigning a value to a new key (that does not exist in the dictionary). For example,

app.py

```
country_capitals = {
  "United States": "New York",
  "Italy": "Naples"
}

# add an item with "Germany" as key and "Berlin" as its value
country_capitals["Germany"] = "Berlin"

print(country_capitals)
# {'United States': 'Washington D.C.',
# 'Italy': 'Rome',
# 'Germany': 'Berlin'}
```

04- Dictionaries in python

Remove Dictionary Items :

We use the del statement to remove an element from the dictionary.

For example,

app.py

```
country_capitals = {
  "United States": "New York",
  "Italy": "Naples"
}

# delete item having "United States" key
del country_capitals["United States"]


print(country_capitals)
# {'Italy': 'Naples'}
```

04- Dictionaries in python

If we need to remove all items from the dictionary at once, we can use the **clear()** method.

app.py

```
country_capitals = {
  "United States": "New York",
  "Italy": "Naples"
}

country_capitals.clear()

print(country_capitals) # {}
```

04- Dictionaries in python

Python Dictionary Methods :

Function	Description
pop()	Remove the item with the specified key.
update()	Add or change dictionary items.
clear()	Remove all the items from the dictionary.
keys()	Returns all the dictionary's keys.
values()	Returns all the dictionary's values.
get()	Returns the value of the specified key.
popitem()	Returns the last inserted key and value as a tuple.
copy()	Returns a copy of the dictionary.

Softy EDUCATION

06 Exercises

python™

05- Exercises

Tuples :

Exercise 1: Swap Elements of Two Tuples

Create a program that takes two tuples, each containing three elements. Your task is to swap the first and last elements of the first tuple with the second and third elements of the second tuple, and vice versa. Return the new tuples.

Exercise 2: Tuple Rotation

Write a function that takes a tuple and a positive integer n as inputs. The function should return a new tuple where the elements are rotated to the right by n positions. For example, if the input tuple is (1, 2, 3, 4) and n is 2, the function should return (3, 4, 1, 2).

05- Exercises

Exercise 3: Frequency of Elements in a Tuple

Given a tuple containing integers, strings, or a mix of data types, write a program that returns a dictionary where the keys are the unique elements from the tuple, and the values are the number of times each element appears in the tuple. Do not use any external libraries or built-in functions that directly solve this task.

05- Exercises

Sets :

Exercise 1: Set Intersection

Write a program that takes two sets of integers from the user and finds and prints the intersection of these sets. (Hint: You may use the & operator or the intersection method.)

Exercise 2: Unique Elements from Lists

Ask the user for two lists of integers and create two sets from them. Write a program that finds and prints the unique elements that are present in the first set but not in the second, and vice versa.

05- Exercises

Exercise 3: Frequency Count

Given a list of integers, write a program that creates a set from the list, then calculates and prints the frequency of each element in the set. You should also display the element in the set with the highest frequency. (Hint: Consider using the Counter from the collections module for frequency counting.)

05- Exercises

Lists :

Exercises 1: Remove Duplicates:

Given a list of integers, create a new list that contains only the unique elements of the original list in the same order they were encountered. Do this without using any built-in Python functions or data structures designed for this purpose.

Exercise 2: Rotate Elements:

Given a list and a number n, rotate the list to the left by n places. For instance, given the list [1, 2, 3, 4, 5] and n = 2, the resulting list should be [3, 4, 5, 1, 2].

05- Exercises

Lists :

Exercises 3: Merge Alternating:

Given two lists of equal length, create a new list that consists of alternating elements from the two lists. For example, given the lists [1, 3, 5] and [2, 4, 6], the resulting list should be [1, 2, 3, 4, 5, 6].

05- Exercises

Dictionaries :

Exercise 1: Dictionary Value Counter

Create a program that takes a dictionary as input, where the values are strings representing categories (e.g., colors, shapes, etc.), and returns a new dictionary with the categories as keys and the counts of occurrences as values.

Exercise 2: Merge Dictionaries

Write a function that takes two dictionaries as input. Both dictionaries have string keys, and the values are lists of numbers. Your function should merge the dictionaries by keys. If a key exists in both dictionaries, combine the lists of numbers. Ensure the numbers in the final lists are unique and sorted in ascending order.

05- Exercises

Dictionaries :

Exercise 3: Nested Dictionary Query

Create a program that handles a nested dictionary, representing a hierarchy of employees in a company. The program should allow users to query an employee's information by name, and then print the information for that employee and all their subordinates (who are also represented by nested dictionaries). If the employee is not found, provide a suitable error message.sure the numbers in the final lists are unique and sorted in ascending order.

Softy EDUCATION

Thank you

python™