Welcome
To
Thinking Programming

Workshop

About Me

- I am Veena A, working as an Assistant Professor, in the Department of Computer Science and Engineering at Dr. Ambedkar Institute of Technology, Bengaluru - 560056.
- Research Interests include Data Science with emphasis on Machine Learning and Deep Learning, and Big Data Analytics
- You can contact me at: **veenaram.cs@drait.edu.in**

Objectives of this Workshop

The objectives of the workshop is to provide hands-on experience in python programming and explore the features of the Python programming language.

To the Audience

- Audience are encouraged to make a note of all the concepts while discussing
- Based on your understanding of the concepts, you need to write the code to solve the questionnaires during Hands-on session
- Go to the following URL if you want to execute the code in parallel while discussing Replace This https://tinyurl.com/DTE-Day3
- Use Audience_Test_the_Code.ipynb file to execute code samples

Creating the Dictionary

The syntax for dictionary is:

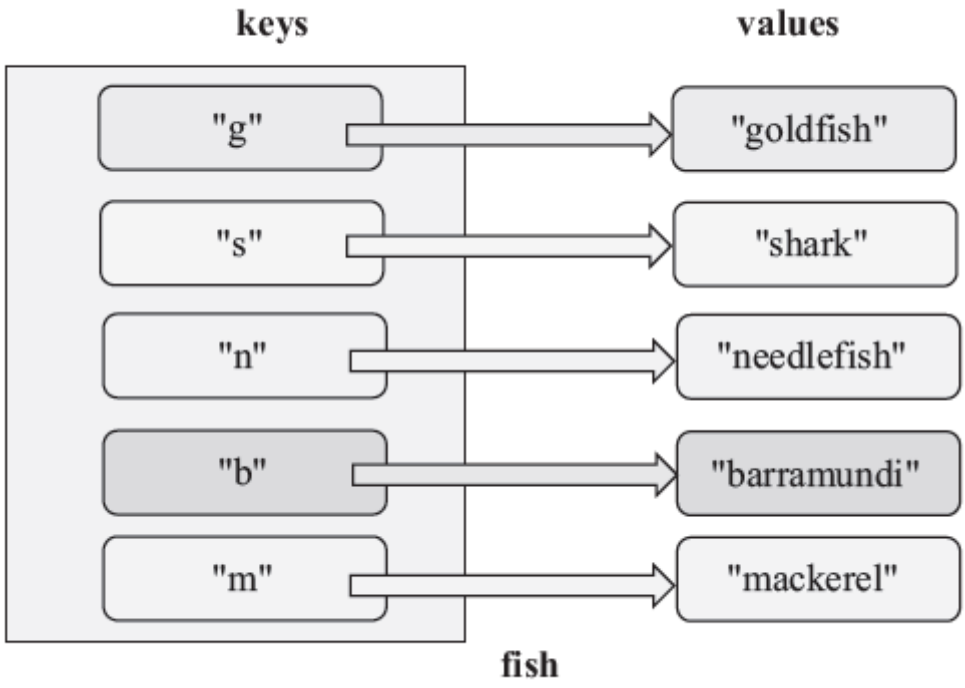
```
dict_variable = {key:value, key:value}
```

In [ ]:

```
#Example for creating the dictionary
```

```
fish = {"g": "goldfish", "s":"shark", "n": "needlefish", "b":"barramundi","m":"macke
fish
```

Out[ ]:
```
{'g': 'goldfish',
 's': 'shark',
 'n': 'needlefish',
 'b': 'barramundi',
 'm': 'mackerel'}
```

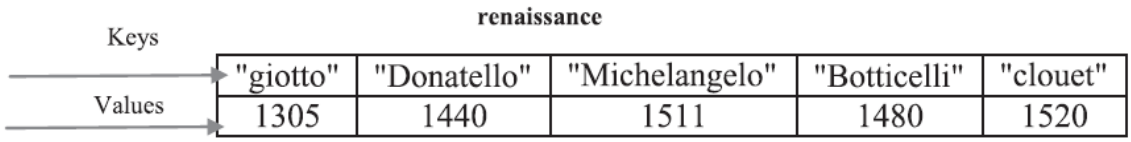


In [ ]:
```
#Keys and Values can have a mixed data type values
mixed_dict = {"portable":"laptop", 9:11, 7:"julius"}
print(type(mixed_dict))
print(mixed_dict)
```

```
<class 'dict'>
{'portable': 'laptop', 9: 11, 7: 'julius'}
```

In [ ]:
```
# Checks if the two dictionaries are equal or not
pizza = {"pepperoni":3, "calzone":5, "margherita":4}
fav_pizza = {"margherita":4, "pepperoni":3, "calzone":5}
pizza == fav_pizza
```

Out[ ]:
```
True
```



In [ ]:
```
#Accessing and Modifying key:value Pairs in Dictionaries
renaissance = {"giotto":1305, "donatello":1440, "michelangelo":1511,"botticelli":148
renaissance
```

Out[ ]:
```
{'giotto': 1305,
 'donatello': 1440,
 'michelangelo': 1511,
```

```
 'botticelli': 1480,
 'clouet': 1520}
```

In [ ]:
```
renaissance["giotto"] = 1310
renaissance
```

Out[ ]:
```
{'giotto': 1310,
 'donatello': 1440,
 'michelangelo': 1511,
 'botticelli': 1480,
 'clouet': 1520}
```

In [ ]:
```
renaissance["leonardo"] = 1503
renaissance
```

Out[ ]:
```
{'giotto': 1310,
 'donatello': 1440,
 'michelangelo': 1511,
 'botticelli': 1480,
 'clouet': 1520,
 'leonardo': 1503}
```

In [ ]:
```
#Name error trying to access the key element which is not present
renaissance["piero"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-39a81c3b9d0e> in <module>
      1 #Name error trying to access the key element which is not present
----> 2 renaissance["piero"]

KeyError: 'piero'
```

In [ ]:
```
#using in keyword
clothes = {"rainy":"raincoats", "summer":"tees", "winter":"sweaters"}
print("spring" in clothes)
print("spring" not in clothes)
```

```
False
True
```

In [ ]:
```
#Using Dict Function to create  Dictionaries
numbers = dict(one=1, two=2, three=3)
numbers
```

Out[ ]:
```
{'one': 1, 'two': 2, 'three': 3}
```

In [ ]:
```
#using the iterables
dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
```

Out[ ]:
```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
```

Built-In Functions Used on Dictionaries

Built-in Functions	Description
len()	The *len()* function returns the number of items (*key:value* pairs) in a dictionary.
all()	The *all()* function returns Boolean True value if all the keys in the dictionary are True else returns False.
any()	The *any()* function returns Boolean True value if any of the key in the dictionary is True else returns False.
sorted()	The *sorted()* function by default returns a list of items, which are sorted based on dictionary keys.

In [ ]:
```
#Built-in Functions used in Dictionaries
presidents = {"washington":1732, "jefferson":1751, "lincoln":1809,"roosevelt":1858,
len(presidents)
```

Out[ ]:
```
5
```

In [ ]:
```
#All function in dict
all_dict_func = {0:True, 2:False}
all(all_dict_func)
```

Out[ ]:
```
False
```

In [ ]:
```
#all method in Dictionary
all_dict_func = {1:True, 2:False}
all(all_dict_func)
```

Out[ ]:
```
True
```

In [ ]:
```
#any method in Dictionary
any_dict_func = {0:True, 2:False}
any(any_dict_func)
```

Out[ ]:
```
True
```

In [ ]:
```
#sorted method in dictionaries
sorted(presidents)
```

Out[ ]:
```
['eisenhower', 'jefferson', 'lincoln', 'roosevelt', 'washington']
```

In [ ]:
```
#sorted method in dictionaries with true
sorted(presidents, reverse = True)
```

Out[ ]:
```
['washington', 'roosevelt', 'lincoln', 'jefferson', 'eisenhower']
```

In [ ]:
```
sorted(presidents.values())
```

Out[ ]:
```
[1732, 1751, 1809, 1858, 1890]
```

In [ ]:
```
sorted(presidents.items())
```

Out[ ]:
```
[('eisenhower', 1890),
 ('jefferson', 1751),
 ('lincoln', 1809),
 ('roosevelt', 1858),
 ('washington', 1732)]
```

Various Dictionary Methods

Dictionary Methods	Syntax	Description
clear()	dictionary_name.clear()	The *clear()* method removes all the *key:value* pairs from the dictionary.
fromkeys()	dictionary_name.fromkeys(seq [, value])	The *fromkeys()* method creates a new dictionary from the given sequence of elements with a value provided by the user.
get()	dictionary_name.get(key [, default])	The *get()* method returns the value associated with the specified key in the dictionary. If the key is not present then it returns the default value. If default is not given, it defaults to *None*, so that this method never raises a *KeyError*.
items()	dictionary_name.items()	The *items()* method returns a new view of dictionary's key and value pairs as tuples.
keys()	dictionary_name.keys()	The *keys()* method returns a new view consisting of all the keys in the dictionary.
pop()	dictionary_name.pop(key[, default])	The *pop()* method removes the key from the dictionary and returns its value. If the key is not present, then it returns the default value. If default is not given and the key is not in the dictionary, then it results in *KeyError*.
popitem()	dictionary_name.popitem()	The *popitem()* method removes and returns an arbitrary (key, value) tuple pair from the dictionary. If the dictionary is empty, then calling *popitem()* results in *KeyError*.
setdefault()	dictionary_name.setdefault (key[, default])	The *setdefault()* method returns a value for the key present in the dictionary. If the key is not present, then insert the key into the dictionary with a default value and return the default value. If key is present, *default* defaults to None, so that this method never raises a *KeyError*.
update()	dictionary_name.update([other])	The *update()* method updates the dictionary with the *key:value* pairs from *other* dictionary object and it returns *None*.
values()	dictionary_name.values()	The *values()* method returns a new view consisting of all the values in the dictionary.

Note: Replace the word "*dictionary_name*" mentioned in the syntax with your *actual dictionary name* in your code.

In [ ]:
```
#Dictionary Methods
box_office_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harrypotter":
box_office_billion_fromkeys = box_office_billion.fromkeys(box_office_billion)
box_office_billion_fromkeys
```

Out[ ]:
```
{'avatar': None,
 'titanic': None,
 'starwars': None,
 'harrypotter': None,
 'avengers': None}
```

In [ ]:
```
#Name Error: frozen key value pair is not present
print(box_office_billion.get("frozen"))
```

```
None
```

In [ ]:
```
box_office_billion.get("frozen",2013)
```

Out[ ]:
```
2013
```

In [ ]:
```
box_office_billion.keys()
```

Out[ ]:
```
dict_keys(['avatar', 'titanic', 'starwars', 'harrypotter', 'avengers'])
```

In [ ]:
```
box_office_billion.values()
```

Out[ ]:
```
dict_values([2009, 1997, 2015, 2011, 2012])
```

In [ ]:
```
box_office_billion.items()
```

Out[ ]:
```
dict_items([('avatar', 2009), ('titanic', 1997), ('starwars', 2015), ('harrypotter',
2011), ('avengers', 2012)])
```

In [ ]:
```
box_office_billion.update({"frozen":2013})
box_office_billion
```

Out[ ]:
```
{'avatar': 2009,
 'titanic': 1997,
 'starwars': 2015,
 'harrypotter': 2011,
 'avengers': 2012,
 'frozen': 2013}
```

In [ ]:
```
box_office_billion.setdefault("minions")
box_office_billion
```

Out[ ]:
```
{'avatar': 2009,
 'titanic': 1997,
 'starwars': 2015,
 'harrypotter': 2011,
 'avengers': 2012,
 'frozen': 2013,
 'minions': None}
```

In [ ]:
```
box_office_billion.pop("avatar")
```

Out[ ]:
```
2009
```

In [ ]:
```
box_office_billion.popitem()
```

Out[ ]:
```
('minions', None)
```

In [ ]:
```
list(box_office_billion.keys())
```

Out[ ]:
```
['titanic', 'starwars', 'harrypotter', 'avengers', 'frozen']
```

In [ ]:
```
list(box_office_billion.values())
```

Out[ ]:
```
[1997, 2015, 2011, 2012, 2013]
```

In [ ]:
```
list(box_office_billion.items())
```

Out[ ]:
```
[('titanic', 1997),
 ('starwars', 2015),
 ('harrypotter', 2011),
```

```
('avengers', 2012),
('frozen', 2013)]
```

In [ ]:
```
box_office_billion.clear()
box_office_billion
```

Out[ ]:
```
{}
```

In [ ]:
```
#Populating Dictionaries with key:value Pairs
countries = {}
countries.update({"Asia":"India"})
countries.update({"Europe":"Germany"})
countries.update({"Africa":"Sudan"})
countries
```

Out[ ]:
```
{'Asia': 'India', 'Europe': 'Germany', 'Africa': 'Sudan'}
```

In [ ]:
```
# Program 7.1: Program to Dynamically Build User Input as a List

def main():
    print("Method 1: Building Dictionaries")
    build_dictionary = {}
    for i in range(0, 2):
        dic_key = input("Enter key ")
        dic_val = input("Enter val ")
        build_dictionary.update({dic_key: dic_val})
    print(f"Dictionary is {build_dictionary}")

    print("Method 2: Building Dictionaries")
    build_dictionary = {}
    for i in range(0, 2):
        dic_key = input("Enter key ")
        dic_val = input("Enter val ")
        build_dictionary[dic_key] = dic_val
    print(f"Dictionary is {build_dictionary}")

    print("Method 3: Building Dictionaries")
    build_dictionary = {}
    i = 0
    while i < 2:
        dict_key = input("Enter key ")
        dict_val = input("Enter val ")
        build_dictionary.update({dict_key: dict_val})
        i = i + 1
    print(f"Dictionary is {build_dictionary}")


if __name__ == "__main__":
    main()
```

```
Method 1: Building Dictionaries
Dictionary is {'name': 'Name', 'vijay': 'name'}
Method 2: Building Dictionaries
Dictionary is {'': ''}
Method 3: Building Dictionaries
Dictionary is {'': ''}
```

In [ ]:
```
# Program 7.2: Program to Illustrate Traversing of key:value Pairs in Dictionaries
# Using for Loop
```

```python
currency = {"india": "rupee", "usa": "dollar", "russia": "ruble", "japan": "yen", "g


def main():
    print("List of Countries")
    for key in currency.keys():
        print(key)

    print("List of Currencies in different Countries")
    for value in currency.values():
        print(value)

    for key, value in currency.items():
        print(f"'{key}' has currency of type '{value}'")


if __name__ == "__main__":
    main()
```

```
List of Countries
india
usa
russia
japan
germany
List of Currencies in different Countries
rupee
dollar
ruble
yen
euro
'india' has currency of type 'rupee'
'usa' has currency of type 'dollar'
'russia' has currency of type 'ruble'
'japan' has currency of type 'yen'
'germany' has currency of type 'euro'
```

In [ ]:

```python
# Program 7.3: Write Python Program to Check for the Presence of a Key in the
# Dictionary and to Sum All Its Values

historical_events = {"apollo11": 1969, "great_depression": 1929, "american_revolutio


def check_key_presence():
    key = input("Enter the key to check for its presence ")
    if key in historical_events.keys():
        print(f"Key '{key}' is present in dictionary")
    else:
        print(f"Key '{key}' is not present in dictionary")


def sum_dictionary_values():
    print("Sum of all the values in the dictionary is")
    print(f"{sum(historical_events.values())}")


def main():
    check_key_presence()
    sum_dictionary_values()


if __name__ == "__main__":
    main()
```

```
Enter the key to check for its presence apollo11
Key 'apollo11' is present in dictionary
Sum of all the values in the dictionary is
7662
```

In [ ]:

```python
# Program 7.4: Write Python Program to Count the Number of Times an Item
# Appears in the List

novels = ["gone_girl", "davinci_code", "games_of_thrones", "gone_girl", "davinci_cod


def main():
    count_items = dict()
    for book_name in novels:
        count_items[book_name] = count_items.get(book_name, 0) + 1

    print("Number of times a novel appears in list is")
    print(count_items)


if __name__ == "__main__":
    main()
```

```
Number of times a novel appears in list is
{'gone_girl': 2, 'davinci_code': 2, 'games_of_thrones': 1}
```

In [ ]:

```python
# Program 7.5: Write Python Program to Count the Number of Times Each Word
# Appears in a Sentence


def main():
    count_words = dict()
    sentence = input("Enter a sentence ")
    words = sentence.split()

    for each_word in words:
        count_words[each_word] = count_words.get(each_word, 0) + 1
    print("The number of times each word appears in a sentence is")
    print(count_words)


if __name__ == "__main__":
    main()
```

```
Enter a sentence This is python Programming
The number of times each word appears in a sentence is
{'This': 1, 'is': 1, 'python': 1, 'Programming': 1}
```

In [ ]:

```python
# Program 7.6: Write Python Program to Count the Number of Characters in a String
# Using Dictionaries. Display the Keys and Their Values in Alphabetical Order


def construct_character_dict(word):
    character_count_dict = dict()
    for each_character in word:
        character_count_dict[each_character] = character_count_dict.get(each_charact

    sorted_list_keys = sorted(character_count_dict.keys())
    for each_key in sorted_list_keys:
        print(each_key, character_count_dict.get(each_key))
```

```
def main():
    word = input("Enter a string ")
    construct_character_dict(word)


if __name__ == "__main__":
    main()
```

```
Enter a string Python Programming
  1
P 2
a 1
g 2
h 1
i 1
m 2
n 2
o 2
r 2
t 1
y 1
```

In [ ]:

```
# Program 7.7: Write Python Program to Generate a Dictionary That Contains (i: i*i)
# Such that i Is a Number Ranging from 1 to n.


def main():
    number = int(input("Enter a number "))
    create_number_dict = dict()
    for i in range(1, number+1):
        create_number_dict[i] = i * i
    print("The generated dictionary of the form (i, i*i) is")
    print(create_number_dict)


if __name__ == "__main__":
    main()
```

```
Enter a number 5
The generated dictionary of the form (i, i*i) is
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

In [ ]:

```
# Program 7.8: Write a Program That Accepts a Sentence and Calculate
# the Number of Digits, Uppercase and Lowercase Letters


def main():
    sentence = input("Enter a sentence ")
    construct_dictionary = {"digits": 0, "lowercase": 0, "uppercase": 0}
    for each_character in sentence:
        if each_character.isdigit():
            construct_dictionary["digits"] += 1
        elif each_character.isupper():
            construct_dictionary["uppercase"] += 1
        elif each_character.islower():
            construct_dictionary["lowercase"] += 1
    print("The number of digits, lowercase and uppercase letters are")
    print(construct_dictionary)


if __name__ == "__main__":
    main()
```

```
Enter a sentence Defective values present
The number of digits, lowercase and uppercase letters are
{'digits': 0, 'lowercase': 21, 'uppercase': 1}
```

In [ ]:

```python
#Program 7.9: The user has to specify a value for n number of students. The Program
#the Registration Number and Marks of a Specified Student Given His Name

def student_details(number_of_students):
    student_name = {}
    for i in range(0, number_of_students):
        name = input("Enter the Name of the Student ")
        registration_number = input("Enter student's Registration Number ")
        total_marks = input("Enter student's Total Marks ")
        student_name[name] = [registration_number, total_marks]
        student_search = input('Enter name of the student you want to search ')
        if student_search not in student_name.keys():
            print('Student you are searching is not present in the class')
        else:
            print("Student you are searching is present in the class")
            print(f"Student's Registration Number is {student_name[student_search][0
            print(f"Student's Total Marks is {student_name[student_search][1]}")

def main():
    number_of_students = int(input("Enter the number of students "))
    student_details(number_of_students)

if __name__ == "__main__":
    main()
```

```
Enter the number of students 2
Enter the Name of the Student Veena
Enter student's Registration Number 1da18PCS01
Enter student's Total Marks Vijay
Enter name of the student you want to search 1da19pcs01
Student you are searching is not present in the class
Enter the Name of the Student Veena
Enter student's Registration Number 44
Enter student's Total Marks 6
Enter name of the student you want to search 7
Student you are searching is not present in the class
```

In [ ]:

```python
# Program 7.10: Program to Demonstrate Nested Dictionaries

student_details = {"name": "jasmine", "registration_number": "1AIT18CS05", "sub_mark


def nested_dictionary():
    print(f"Student Name {student_details['name']}")
    print(f"Registration Number {student_details['registration_number']}")
    average = sum(student_details["sub_marks"].values()) / len(student_details["sub_
    print(f"Average of all the subjects is {average}")


def main():
    nested_dictionary()


if __name__ == "__main__":
    main()
```

```
Student Name jasmine
```

```
Registration Number 1AIT18CS05
Average of all the subjects is 90.0
```

In [ ]:
```
animals = {"r":"raccoon", "c":"cougar", "m":"moose"}
animals
```

Out[ ]:
```
{'r': 'raccoon', 'c': 'cougar', 'm': 'moose'}
```

In [ ]:
```
del animals["c"]
animals
```

Out[ ]:
```
{'r': 'raccoon', 'm': 'moose'}
```

Programming Excercise

The information is given as a table of state names (keys) and the population of each state (values). The program should allow the user to enter the name of a state. If the state is found in the dictionary, then the program should report the population of that state. If the state is not found, then the program should output a message like, "Sorry, but we do not have information for that state." The program should run in a loop and allow the user to enter any number of states, and then exit when the user presses Enter.

State	Population
Karnataka	38802000
Manipur	26956000
Gujarat	19893000
Jammu and Kashmir	19746000
Andhra Pradesh	12880000
Mizoram	12787000
Delhi	11594000
Madhya Pradesh	10097000
Maharashtra	26956000
Kerala	9909000
Tamil Nadu	8938000

In [ ]:
```
# Get the population of a given state
statesDict = {
'Karnataka':38802000, 'Manipur':26956000, 'Gujarat':19893000, 'Jammu and Kashmir':19
'Andhra Pradesh': 12880000, 'Mizoram': 12787000, 'Delhi':11594000, 'Madhya Pradesh':
'Maharashtra': 9943964, 'Kerala':9909000, 'Tamil Nadu': 8938000}
while True:
    usersState = input('Enter a state: ')
    if usersState == '':
        break
    if usersState in statesDict:
        population = statesDict[usersState]
        print('The population of', usersState, 'is', population)
    else:
        print('Sorry, but we do not have any information about', usersState)
```

```
The population of Karnataka is 38802000
```

Example of using a dictionary is a program that works as a real dictionary; in this example, a dictionary of a few of the programming terms introduced in this book. In the following program, programming terms are used as keys and their matching definitions are specified as values.

In [ ]:

```
programmingDict = {
'variable': 'A named memory location that holds a value',
'loop' : 'A block of code that is repeated until a certain condition is met.',
'function' : 'A series of related steps that form a larger task, often called from m
'constant' : 'A variable whose value does not change',
'Boolean' : 'A data type that can only have values of True or False'}
while True:
    usersWord = input('Enter a word to look up (or Return to quit): ')
    if usersWord == '':
        break
    if usersWord in programmingDict:
        definition = programmingDict[usersWord]
        print('The definition of', usersWord, 'is:')
        print(definition)
    else:
        print('The word', usersWord, 'is not in our dictionary.')
        yesOrNo = input('Would you like to add a definition for ' + usersWord + ' (y
        if yesOrNo.lower() == 'y':
            usersDefinition = input('Please give a definition for ' + usersWord + ':
            programmingDict[usersWord] = usersDefinition
            print('Thanks, got it!')
            print('Done.')
            print(programmingDict)
```

```
The word Veena is not in our dictionary.
Thanks, got it!
Done.
{'variable': 'A named memory location that holds a value', 'loop': 'A block of code
that is repeated until a certain condition is met.', 'function': 'A series of relate
d steps that form a larger task, often called from multiple places in a program', 'c
onstant': 'A variable whose value does not change', 'Boolean': 'A data type that can
only have values of True or False', 'Veena': 'Name of a person'}
```

In [ ]:

```
def main() :
    myContacts = {"Fred": 7235591, "Mary": 3841212,"Bob": 3841212, "Sarah": 2213278
     # See if Fred is in the list of contacts.
    if "Fred" in myContacts :
        print("Number for Fred:", myContacts["Fred"])
    else :
        print("Fred is not in my contact list.")

 # Get and print a list of every contact with a given number.
    nameList = findNames(myContacts, 3841212)
    print("Names for 384-1212: ", end="")
    for name in nameList :
        print(name, end=" ")
        print()

    # Print a list of all names and numbers.
    printAll(myContacts)

 ## Find all names associated with a given telephone number.
 # @param contacts the dictionary
 # @param number the telephone number to be searched
 # @return the list of names
```

```
 #
def findNames(contacts, number) :
    nameList = []
    for name in contacts :
        if contacts[name] == number :
            nameList.append(name)
    return nameList

 ## Print an alphabetical listing of all dictionary items.
 # @param contacts the dictionary
 #
def printAll(contacts) :
    print("All names and numbers:")
    for key in sorted(contacts) :
        print("%-10s %d" % (key, contacts[key]))

 # Start the program.
main()
```

```
Number for Fred: 7235591
Names for 384-1212: Mary
Bob
All names and numbers:
Bob        3841212
Fred       7235591
Mary       3841212
Sarah      2213278
```

Write a program to take in the roll number, name and percentage of marks for n students of Class X and do the following:

1. Accept details of the n students (n is the number of students).
2. Search details of a particular student on the basis of roll number and display result.
3. Display the result of all the students.
4. Find the topper amongst them.
5. Find the subject toppers amongst them. **(Hint: Use Dictionary, where the key can be roll number and the value an immutable data type containing name and percentage.)**

Our heritage monuments are our assets. They are a reflection of our rich and glorious past and an inspiration for our future. UNESCO has identified some of Indian heritage sites as World Heritage sites. Collect the following information about these sites:

1. What is the name of the site?
2. Where is it located?
3. District
4. State
5. When was it built?
6. Who built it?
7. Why was it built?
8. Website link (if any) Write a Python program to: • add, modify or delete an entered heritage site in the list of sites. • Display the list of world heritage sites in India. • Search and display information of a world heritage site entered by the user

Case Study Program to find the winner of the election

In [ ]:

```
# Function to find winner of an election where votes
# are represented as candidate names
```

```python
from collections import Counter

def winner(input):

    # convert list of candidates into dictionary
    # output will be likes candidates = {'A':2, 'B':4}
    votes = Counter(input)

    # create another dictionary and it's key will
    # be count of votes values will be name of
    # candidates
    dict = {}

    for value in votes.values():

        # initialize empty list to each key to
        # insert candidate names having same
        # number of votes
        dict[value] = []

    for (key,value) in votes.items():
        dict[value].append(key)

    # sort keys in descending order to get maximum
    # value of votes
    maxVote = sorted(dict.keys(),reverse=True)[0]

    # check if more than 1 candidates have same
    # number of votes. If yes, then sort the list
    # first and print first element
    print("The candidate who has the maximum number of votes is")
    if len(dict[maxVote])>1:
        print (sorted(dict[maxVote])[0])
    else:
        print (dict[maxVote][0])

# Driver program
if __name__ == "__main__":
    input =['john','johnny','jackie','johnny',
            'john','jackie','jamie','jamie',
            'john','johnny','jamie','johnny',
            'john']
    winner(input)
```