**1. Python theory questions**

1. What is Python and what are its main features?

Python is a high-level, interpreted, general-purpose programming language.

Main features: easy to code; free and open source; object-oriented language; high-level language; extensible feature; python is portable language; interpreted language; dynamically typed language.

2. Discuss the difference between Python 2 and Python 3

Python 3 syntax is simpler and easily understandable whereas Python 2 syntax is comparatively difficult to understand.

Python 3 default storing of strings is Unicode whereas Python 2 stores need to define Unicode string value with “u.”

Python 3 value of variables never changes whereas in Python 2 value of the global variable will be changed while using it inside for-loop.

Python 3 exceptions should be enclosed in parenthesis while Python 2 exceptions should be enclosed in notations.

Python 3 rules of ordering comparisons are simplified whereas Python 2 rules of ordering comparison are complex.

Python 3 offers Range() function to perform iterations whereas, In Python 2, the xrange() is used for iterations.

3. What is PEP 8?

It’s a document that provides guidelines and best practices on how to write Python code. It was written in 2001 by Guido van Rossum, Barry Warsaw, and Nick Coghlan. The primary focus of PEP 8 is to improve the readability and consistency of Python code.

4. In computing / computer science what is a program?

A program is a specific set of ordered operations for a computer to perform. Typically, the program is put into a storage area accessible to the computer.

5. In computing / computer science what is a process?

A process is an instance of a program running in a computer. It is close in meaning to task , a term used in some operating systems.

6. In computing / computer science what is cache?

Caches are used to store temporary files, using hardware and software components

7. In computing / computer science what is a thread and what do we mean by

multithreading?

A thread is a small set of instructions designed to be scheduled and executed by the CPU independently of the parent process. For example, a program may have an open thread waiting for a specific event to occur or running a separate job, allowing the main program to perform other tasks.

Multithreading is the ability of a program or an operating system to enable more than one user at a time without requiring multiple copies of the program running on the computer. Multithreading can also handle multiple requests from the same user.

8. In computing / computer science what is concurrency and parallelism and what are the differences?

Concurrency relates to an application that is processing more than one task at the same time. Concurrency is an approach that is used for decreasing the response time of the system by using the single processing unit.

Parallelism is related to an application where  tasks are divided into smaller sub-tasks that are processed seemingly simultaneously or parallel. It is used to increase the throughput and computational speed of the system by using multiple processors.

Differences:

1.Concurrency is the task of running and managing multiple computations at the same time. While parallelism is the task of running multiple computations simultaneously.

2.Concurrency is achieved through the interleaving operation of processes on the central processing unit(CPU) or in other words by the context switching. While it is achieved by through multiple central processing units(CPUs).

3.Concurrency can be done by using a single processing unit. While this can’t be done by using a single processing unit. it needs multiple processing units.

4.Concurrency increases the amount of work finished at a time. While it improves the throughput and computational speed of the system.

5.Concurrency deals lot of things simultaneously. While it do lot of things simultaneously.

6.Concurrency is the non-deterministic control flow approach. While it is deterministic control flow approach.

7.In concurrency debugging is very hard. While in this debugging is also hard but simple than concurrency.

9. What is GIL in Python and how does it work?

The Python Global Interpreter Lock (GIL) is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter. This means that only one thread can be in a state of execution at any point in time.

10. What do these software development principles mean: DRY, KISS, BDUF

The DRY (don't repeat yourself) principle is a best practice in software development that recommends software engineers to do something once, and only once.

The KISS (Keep it simple, stupid) is a design principle which states that designs and/or systems should be as simple as possible. Wherever possible, complexity should be avoided in a system—as simplicity guarantees the greatest levels of user acceptance and interaction.

The BDUF (Big Design Up Front) is a software development approach in which the program's design is to be completed and perfected before that program's implementation is started. It is often associated with the waterfall model of software development.

11. What is a Garbage Collector in Python and how does it work?

The garbage collector is keeping track of all objects in memory. Python garbage collector helps you avoid memory leaks by detecting circular references and destroy objects appropriately.

How it works:

Keep your Python objects being referenced, or they will be released in memory. The Python garbage collector initiates its execution with the program and is activated if the reference count falls to zero.

When we assign the new name or placed it in containers such as a dictionary or tuple, the reference count increases its value. If we reassign the reference to an object, the reference counts decreases its value if. It also decreases its value when the object's reference goes out of scope or an object is deleted.

12. How is memory managed in Python?

Memory management in Python involves a private heap containing all Python objects and data structures. The management of this private heap is ensured internally by the Python memory manager.

13. What is a Python module?

What are modules in Python? Modules refer to a file containing Python statements and definitions. A file containing Python code, for example: example.py , is called a module, and its module name would be example . We use modules to break down large programs into small manageable and organized files.

14. What is docstring in Python?

A Python docstring is a string used to document a Python module, class, function or method, so programmers can understand what it does without having to read the details of the implementation. Also, it is a common practice to generate online (html) documentation automatically from docstrings.

15. What is pickling and unpickling in Python? Example usage.

“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

Example:

import pickle

favorite\_color = { "student": "yellow", "teacher": "red" }

pickle.dump( favorite\_color, open( "save.p", "wb" ) )

import pickle

favorite\_color = pickle.load( open( "save.p", "rb" ) )

# favorite\_color is now { "student": "yellow", "teacher": "red" }

16. What are the tools that help to find bugs or perform static analysis?

Pychecker and Pylint are the static analysis tools that help to find bugs in python. Pychecker is an opensource tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.

17. How are arguments passed in Python by value or by reference? Give an example.

If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like Call-by-value. It's different, if we pass mutable arguments. All parameters (arguments) in the Python language are passed by reference.

Example of by value:

def change(data):

data=45

drint(“Inside Function:”, data)

def main():

data=20

print(“Before Calling :”, data)

change(data)

print(“After Calling:”, data)

if \_name\_==”\_main\_”:main()

This will output:

Before Calling : 20

Inside Function: 45

After Calling : 20

Example of by reference:

def change(data):

data[1]=25

print(“Inside Function :”,data)

def main():

data=[10,20,30,40,50]

print(“Before Calling :”,data)

change(data)

print(“After Calling :”, data)

if \_name\_==”\_main\_”:main()

This will output:  
Before Calling : [10,20,30,40,50]

Inside Calling : [10,25,30,40,50]

After Calling : [10, 25, 30, 40,50]

18. What are Dictionary and List comprehensions in Python? Provide examples.

List comprehensions and dictionary comprehensions are a powerful substitute to for-loops and also lambda functions. Not only do list and dictionary comprehensions make code more concise and easier to read, they are also faster than traditional for-loops. It allows us to loop through existing data and filter it to create a list or dictionary, or we can even loop through a sequence that we generate and make a list or dictionary from that sequence.

Example of list comprehension:

h\_letters = [ letter for letter in ‘human’]

print(h\_letters)

It will output:

[‘h’, ‘u’, ‘m’, ‘a’, ‘n’]

Example of dictionary comprehension:

h\_words=[‘human’, ‘man’, ‘woman’]

p\_words={w:len(w) for w in h\_words}

print(p\_words)

It will output:

{‘human’:5, ‘man’:3, ‘woman’:5}

19. What is namespace in Python?

A namespace is a collection of currently defined symbolic names along with information about the object that each name references. You can think of a namespace as a dictionary in which the keys are the object names and the values are the objects themselves.

20.What is pass in Python?

The pass statement is used as a null statement, a placeholder for future code. When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

21. What is unit test in Python?

A unit test is a test that checks a single component of code, usually modularized as a function, and ensures that it performs as expected. Unit tests are an important part of regression testing to ensure that the code still functions as expected after making changes to the code and helps ensure code stability.

22. In Python what is slicing?

Slicing in Python is a feature that enables accessing parts of sequences like strings, tuples, and lists. You can also use them to modify or delete the items of mutable sequences such as lists.

23. What is a negative index in Python?

This means the last value of a sequence has an index of -1, the second last -2, and so on. You can use negative indexing as your advantage when you want to pick values from the end (right side) of an iterable.

24.How can the ternary operators be used in python? Give an example.

Ternary operators are also known as conditional expressions are operators that evaluate something based on a condition being true or false. It allows testing a condition in a single line replacing the multiline if-else making the code compact.

Example:

pass\_the\_exam = True

print(’Very good’ if pass\_the\_exam else ‘Bad’)

Output:

Very good

25. What does this mean: \*args, \*\*kwargs? And why would we use it?

Python has \*args which allow us to pass the variable number of non keyword arguments to function.

\*\*kwargs allows us to pass the variable length of keyword arguments to the function.

\*args is to solve the problem that we are not sure about the number of arguments that can be passed to a function.

\*\*kwargs is used because Python passes variable length non keyword argument to function using \*args but we cannot use this to pass keyword argument.

26. How are range and xrange different from one another?

For the most part, xrange and range are the exact same in terms of functionality. They both provide a way to generate a list of integers for you to use, however you please. The only difference is that range returns a Python list object and xrange returns an xrange object.

27. What is Flask and what can we use it for?

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file.

28. What are clustered and non-clustered index in a relational database?

A Clustered index is a type of index in which table records are physically reordered to match the index. A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk.

29. What is a ‘deadlock’ a relational database?

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks. For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish.

30.What is a ‘livelock’ a relational database?

A Livelock is a situation where a request for an exclusive lock is denied repeatedly, as many overlapping shared locks keep on interfering each other. The processes keep on changing their status, which further prevents them from completing the task. This further prevents them from completing the task.

**2. Python string methods:**

capitalize()

The capitalize() method converts the first character of a string to an uppercase letter and all other alphabets to lowercase.

sentence = ‘i love YOU’

capitalized\_string = sentence.capitalize()

print(capitalized\_string)

Output: I love you

casefold()

The casefold() method converts all characters of the string into lowercase letters and returns a new string.

text = "pYtHon"

lowercased\_string = text.casefold()

print(lowercased\_string)

Output: python

center()

The center() method returns a new centered string after padding it with the specified character.

sentence = "Python is awesome"

new\_string = sentence.center(24, '\*')

print(new\_string)

Output: \*\*\*Python is awesome\*\*\*\*

count()

The count() method returns the number of occurrences of a substring in the given string.

message = 'python is popular programming language'

print('Number of occurrence of p:', message.count('p'))

Output: Number of occurrence of p: 4

endswith()

The endswith() method returns True if a string ends with the specified suffix. If not, it returns False.

message = 'Python is fun'

print(message.endswith('fun'))

Output: True

find()

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

message = 'Python is a fun programming language'

print(message.find('fun'))

Output: 12

format()

The string format() method formats the given string into a nicer output in Python.

print("Hello {}, your balance is {}.".format("Adam", 230.2346))

Output: Hello Adam, your balance is 230.2346.

index()

The index() method returns the index of a substring inside the string (if found). If the substring is not found, it raises an exception.

text = 'Python is fun'

result = text.index('is')

print(result)

Output: 7

isalnum()

The isalnum() method returns True if all characters in the string are alphanumeric (either alphabets or numbers). If not, it returns False.

name1 = "Python3"

print(name1.isalnum())

Output: True

name2 = "Python 3"

print(name2.isalnum())

Output: False

isalpha()

The isalpha() method returns True if all characters in the string are alphabets. If not, it returns False.

name = "Monica"

print(name.isalpha())

name = "Monica Geller"

print(name.isalpha())

name = "Mo3nicaGell22er"

print(name.isalpha())

Output:

True

False

False

isdigit()

The isdigit() method returns True if all characters in a string are digits. If not, it returns False.

str1 = '342'

print(str1.isdigit())

str2 = 'python'

print(str2.isdigit())

Output: True

False

islower()

The islower() method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.

s = 'this is good'

print(s.islower())

s = 'th!s is a1so g00d'

print(s.islower())

s = 'this is Not good'

print(s.islower())

Output:

True

True

False

isnumeric()

The isnumeric() method returns True if all characters in a string are numeric characters. If not, it returns False.

s = '1242323'

print(s.isnumeric())

s = '²3455'

s = '\u00B23455'

print(s.isnumeric())

s = '½'

s = '\u00BD'

print(s.isnumeric())

s = '1242323'

s='python12'

print(s.isnumeric())

Output

True

True

True

False

isspace()

The isspace() method returns True if there are only whitespace characters in the string. If not, it return False.

s = ' \t'

print(s.isspace())

s = ' a '

print(s.isspace())

s = ''

print(s.isspace())

Output:

True

False

False

istitle()

The istitle() returns True if the string is a titlecased string. If not, it returns False.

s = 'PYTHON'

print(s.istitle())

Output:False

isupper()

The string isupper() method returns whether or not all characters in a string are uppercased or not.

string = "THIS IS not GOOD!"

print(string.isupper())

Output:False

join()

The join() string method returns a string by joining all the elements of an iterable (list, string, tuple), separated by a string separator.

text = ['Python', 'is', 'a', 'fun', 'programming', 'language']

print(' '.join(text))

Output: Python is a fun programming language

lower()

The lower() method converts all uppercase characters in a string into lowercase characters and returns it.

message = 'PYTHON IS FUN'

print(message.lower())

Output: python is fun

lstrip()

The lstrip() method returns a copy of the string with leading characters removed (based on the string argument passed).

random\_string = ' this is good '

print(random\_string.lstrip())

print(random\_string.lstrip('sti'))

print(random\_string.lstrip('s ti'))

Output:

this is good

this is good

his is good

replace()

The replace() method replaces each matching occurrence of the old character/text in the string with the new character/text.

text = 'bat ball'

replaced\_text = text.replace('b', 'c')

print(replaced\_text)

Output: cat call

rsplit()

The rsplit() method splits string from the right at the specified separator and returns a list of strings.

text= 'Love thy neighbor'

print(text.rsplit())

Output: [‘Love’, ‘thy’, ‘neighbor’]

rstrip()

The rstrip() method returns a copy of the string with trailing characters removed (based on the string argument passed).

title = 'Python Programming '

result = title.rstrip()

print(result)

Output: Python Programming

split()

The split() method breaks up a string at the specified separator and returns a list of strings.

text = 'Python is a fun programming language'

print(text.split(' '))

Output: ['Python', 'is', 'a', 'fun', 'programming', 'language']

splitlines()

The splitlines() method splits the string at line breaks and returns a list of lines in the string.

grocery = 'Milk\nChicken\r\nBread\rButter'

print(grocery.splitlines())

Output: ['Milk', 'Chicken', 'Bread', 'Butter']

startswith()

The startswith() method returns True if a string starts with the specified prefix(string). If not, it returns False.

message = 'Python is fun'

print(message.startswith('Python'))

Output: True

strip()

The strip() method returns a copy of the string by removing both the leading and the trailing characters (based on the string argument passed).

message = ' Learn Python '

print('Message:', message.strip())

Output: Message: Learn Python

swapcase()

The string swapcase() method converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it.

string = "ThIs ShOuLd Be MiXeD cAsEd."

print(string.swapcase())

Output: tHiS sHoUlD bE mIxEd CaSeD.

title()

The title() method returns a string with first letter of each word capitalized; a title cased string.

text = 'My favorite number is 25.'

print(text.title())

Output: My Favorite Number Is 25.

upper()

The upper() method converts all lowercase characters in a string into uppercase characters and returns it.

message = 'python is fun'

print(message.upper())

Output: PYTHON IS FUN

**3. Python list methods:**

append()

The append() method adds an item to the end of the list.

currencies = ['Dollar', 'Euro', 'Pound']

currencies.append('Yen')

print(currencies)

Output: ['Dollar', 'Euro', 'Pound', 'Yen']

clear()

The clear() method removes all items from the list.

prime\_numbers = [2, 3, 5, 7, 9, 11]

prime\_numbers.clear()

print('List after clear():', prime\_numbers)

Output: List after clear(): []

copy()

The copy() method returns a shallow copy of the list.

prime\_numbers = [2, 3, 5]

numbers = prime\_numbers.copy()

print('Copied List:', numbers)

Output: Copied List: [2, 3, 5]

count()

The count() method returns the number of times the specified element appears in the list.

numbers = [2, 3, 5, 2, 11, 2, 7]

count = numbers.count(2)

print('Count of 2:', count)

Output: Count of 2: 3

extend()

The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

prime\_numbers = [2, 3, 5]

numbers = [1, 4]

numbers.extend(prime\_numbers)

print('List after extend():', numbers)

Output: List after extend(): [1, 4, 2, 3, 5]

index()

The index() method returns the index of the specified element in the list.

animals = ['cat', 'dog', 'rabbit', 'horse']

index = animals.index('dog')

print(index)

Output: 1

insert()

The insert() method inserts an element to the list at the specified index.

vowel = ['a', 'e', 'i', 'u']

vowel.insert(3, 'o')

print('List:', vowel)

Output: List: ['a', 'e', 'i', 'o', 'u']

pop()

The pop() method removes the item at the given index from the list and returns the removed item.

prime\_numbers = [2, 3, 5, 7]

removed\_element = prime\_numbers.pop(2)

print('Removed Element:', removed\_element)

print('Updated List:', prime\_numbers)

Output:

Removed Element: 5

Updated List: [2, 3, 7]

remove()

The remove() method removes the first matching element (which is passed as an argument) from the list.

prime\_numbers = [2, 3, 5, 7, 9, 11]

prime\_numbers.remove(9)

print('Updated List: ', prime\_numbers)

Output: Updated List: [2, 3, 5, 7, 11]

reverse()

The reverse() method reverses the elements of the list.

prime\_numbers = [2, 3, 5, 7]

prime\_numbers.reverse()

print('Reversed List:', prime\_numbers)

Output: Reversed List: [7, 5, 3, 2]

sort()

The sort() method sorts the items of a list in ascending or descending order.

prime\_numbers = [11, 3, 7, 5, 2]

prime\_numbers.sort()

print(prime\_numbers)

Output: [2, 3, 5, 7, 11]

**4. Python tuple methods:**

count()

The count() method returns the number of times the specified element appears in the tuple.

vowels = ('a', 'e', 'i', 'o', 'i', 'u')

count = vowels.count('i')

print('The count of i is:', count)

Output:

The count of i is: 2

index()

The index() method returns the index of the specified element in the tuple.

vowels = ('a', 'e', 'i', 'o', 'i', 'u')

index = vowels.index('e')

print('The index of e:', index)

Output: The index of e: 1

**5. Python dictionary methods:**

clear()

The clear() method removes all items from the dictionary.

d = {1: "one", 2: "two"}

d.clear()

print('d =', d)

Output

d = {}

copy()

They copy() method returns a copy (shallow copy) of the dictionary.

original\_marks = {'Physics':67, 'Maths':87}

copied\_marks = original\_marks.copy()

print('Original Marks:', original\_marks)

print('Copied Marks:', copied\_marks)

Output: Original Marks: {'Physics': 67, 'Maths': 87}

Copied Marks: {'Physics': 67, 'Maths': 87}

fromkeys()

The fromkeys() method creates a new dictionary from the given sequence of elements with a value provided by the user.

keys = {'a', 'e', 'i', 'o', 'u' }

value = 'vowel'

vowels = dict.fromkeys(keys, value)

print(vowels)

Output

{'a': 'vowel', 'u': 'vowel', 'o': 'vowel', 'e': 'vowel', 'i': 'vowel'}

get()

The get() method returns the value for the specified key if the key is in the dictionary.

marks = {'Physics':67, 'Maths':87}

print(marks.get('Physics'))

Output: 67

items()

The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

marks = {'Physics':67, 'Maths':87}

print(marks.items())

Output: dict\_items([('Physics', 67), ('Maths', 87)])

keys()

The keys() method returns a view object that displays a list of all the keys in the dictionary.

person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}

print(person.keys())

empty\_dict = {}

print(empty\_dict.keys())

Output:

dict\_keys(['name', 'salary', 'age'])

dict\_keys([])

pop()

The pop() method removes and returns an element from a dictionary having the given key.

marks = { 'Physics': 67, 'Chemistry': 72, 'Math': 89 }

element = marks.pop('Chemistry')

print('Popped Marks:', element)

Output: Popped Marks: 72

popitem()

The Python popitem() method removes and returns the last element (key, value) pair inserted into the dictionary.

person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}

result = person.popitem()

print('Return Value = ', result)

print('person = ', person)

Output:

Return Value = ('salary', 3500.0)

person = {'name': 'Phill', 'age': 22}

setdefault()

The setdefault() method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary.

person = {'name': 'Phill', 'age': 22}

age = person.setdefault('age')

print('person = ',person)

print('Age = ',age)

Output

person = {'name': 'Phill', 'age': 22}

Age = 22

update()

The update() method updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.

marks = {'Physics':67, 'Maths':87}

internal\_marks = {'Practical':48}

marks.update(internal\_marks)

print(marks)

Output: {'Physics': 67, 'Maths': 87, 'Practical': 48}

values()

The values() method returns a view object that displays a list of all the values in the dictionary.

marks = {'Physics':67, 'Maths':87}

print(marks.values())

Output: dict\_values([67, 87])

**6. Python set methods:**

add()

The add() method adds a given element to a set. If the element is already present, it doesn't add any element.

prime\_numbers = {2, 3, 5, 7}

prime\_numbers.add(11)

print(prime\_numbers)

Output: {2, 3, 5, 7, 11}

clear()

The clear() method removes all elements from the set.

vowels = {'a', 'e', 'i', 'o', 'u'}

print('Vowels (before clear):', vowels)

vowels.clear()

print('Vowels (after clear):', vowels)

Output:

Vowels (before clear): {'e', 'a', 'o', 'u', 'i'}

Vowels (after clear): set()

copy()

The copy() method returns a shallow copy of the set.

numbers = {1, 2, 3, 4}

new\_numbers = numbers.copy()

new\_numbers.add(5)

print('numbers: ', numbers)

print('new\_numbers: ', new\_numbers)

Output:

numbers: {1, 2, 3, 4}

new\_numbers: {1, 2, 3, 4, 5}

difference()

The difference() method returns the set difference of two sets.

A = {'a', 'b', 'c', 'd'}

B = {'c', 'f', 'g'}

print(A.difference(B))

print(B.difference(A))

Output:

{'b', 'a', 'd'}

{'g', 'f'}

intersection()

The intersection() method returns a new set with elements that are common to all sets.

A = {2, 3, 5}

B = {1, 3, 5}

print(A.intersection(B))

Output: {3, 5}

issubset()

The issubset() method returns True if all elements of a set are present in another set (passed as an argument). If not, it returns False.

A = {1, 2, 3}

B = {1, 2, 3, 4, 5}

print(A.issubset(B))

Output:True

issuperset()

The issuperset() method returns True if a set has every elements of another set (passed as an argument). If not, it returns False.

A = {1, 2, 3, 4, 5}

B = {1, 2, 3}

print(A.issuperset(B))

print(B.issuperset(A))

Output:

True

False

pop()

The pop() method removes an arbitrary element from the set and returns the element removed.

A ={'a', 'b', 'c', 'd'}

print('Return Value is', A.pop())

Output:

Return Value is d

remove()

The remove() method removes the specified element from the set.

languages = {'Python', 'Java', 'English'}

languages.remove('English')

print(languages)

Output: {'Python', 'Java'}

symmetric\_difference()

The Python symmetric\_difference() method returns the symmetric difference of two sets.

A = {'a', 'b', 'c', 'd'}

B = {'c', 'd', 'e' }

print(A.symmetric\_difference(B))

print(B.symmetric\_difference(A))

Output:

{'b', 'a', 'e'}

{'b', 'e', 'a'}

union()

The Python set union() method returns a new set with distinct elements from all the sets.

A = {2, 3, 5}

B = {1, 3, 5}

print('A U B = ', A.union(B))

Output: A U B = {1, 2, 3, 5}

update()

The Python set update() method updates the set, adding items from other iterables.

string\_alphabet = 'abc'

numbers\_set = {1, 2}

numbers\_set.update(string\_alphabet)

print('numbers\_set =', numbers\_set)

info\_dictionary = {'key': 1, 'lock' : 2}

numbers\_set = {'a', 'b'}

numbers\_set.update(info\_dictionary)

print('numbers\_set =', numbers\_set)

Output:

numbers\_set = {'c', 1, 2, 'b', 'a'}

numbers\_set = {'key', 'b', 'lock', 'a'}

**7. Python file methods:**

read()

The read() method returns the specified number of bytes from the file.

In demo.txt, there are

Testing - FirstLine

Testing - SecondLine

Testing - Third Line

f = open("demo.txt", "r")

print(f.read(7))

f.close()

Output:

Testing

readline()

Python readline() is a file method that helps to read one complete line from the given file.

f = open("demo.txt", "r")

print(f.readline())

f.close()

Output:

Testing - FirstLine

readlines()

The readlines() method returns a list containing each line in the file as a list item.

f = open("demo.txt", "r")

print(f.readlines())

f.close()

Output:

[Testing - FirstLine

Testing - SecondLine

Testing - Third Line]

write()

The write() method writes a specified text to the file.

f = open("demo2.txt", "a")

f.write("Now the file has more content!")

f.close()

writelines()

The writelines() method writes the items of a list to the file.

f = open("demofile3.txt", "a")

f.writelines(["See you soon!", "Over and out."])

f.close()