THEORY QUESTIONS ASSIGNMENT

Python based theory

To be completed at student's own pace and submitted before given deadline

NO	TASK	POINTS
PYTHON		
1	Theory questions	30
2	String methods	29
3	List methods	11
4	Dictionary methods	11
5	Tuple methods	2
6	Set methods	12
7	File methods	5
	TOTAL	**100**

1. Python theory questions	**30 points**

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

   Python is a free open-source dynamic high level programming language which uses object orientated programming and has a large standard library which give you access to the built-in functions and high-level data structures.

2. Discuss the difference between Python 2 and Python 3

   Python 3 helps to change the way it handles strings to make everything easier instead of using Python 2.

   The print is handled differently in Python 3 vs Python 2 as print is treated as a function where you pass the parameters into the parenthesis and if not done this will provide a syntax error. In Python 2 print is a statement so you do not surround the parameters in parenthesis.

   During division Python 2 will round the number to the integer where here it will round down to give 1 when 3/2 whereas Python 3 will give the float 1.5 as it reads 3.0/2.0.

Python 3 by default stores the strings as Unicode, but in Python 2 the string needs you to mark itself with a u if you want to store it by Unicode as by default the strings use ASCII. Unicode is more versatile than ASCII and therefore is preferred to use.

3. What is PEP 8?

   It is a document written by Guido van Rossum, Barry Warsaw, and Nick Coghlan in 2001 the gives you the best guidelines and practises when writing code on Python. The aim of this document is to help improve the readability and consistency when writing code in Python.

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

   This is an executable software that runs on a computer that is made up from compiled code that contains a specific set of ordered operations that can run directly from the operating system on the computer For example web browsers like Google Chrome or Firefox.

5. In computing / computer science what is a process?
   A process is an instance of a computer program that is implemented by one or more than one thread and runs on the computer system where it uses its resources and is a program or part of a programme.

6. In computing / computer science what is cache?
   Cache is known as a small amount of memory which is part of the central processing unit CPU where the more cache there is the more data date can be stored in a closer proximity to the CPU. A cache is used to help improve the performance of a processor and helps to access data more frequently due to its high-speed memory.

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

   A single thread is a light-weight sub process of the smallest unit which contains a self-contained sequence of instructions in the CPU to execute the instructions set. Multi-threading is a CPU feature which allows two or more threads to be formed and run independently where they are sharing the process resources. Whilst executing multiple threads from the CPU they are able to run parallel to let concurrent tasks happen at the same time similar to running multiple programs at the same time as the threads are able to share information and communicate with each other.

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

   Concurrency is the way that processes run and is the ability of the application to make progress on more than on task at the same time so it is able to handle and not actually complete many tasks. If the computer has one CPU then the application will can make progress on multiple tasks that are being processed at the same time but not simultaneously. So one task is not completely finished before starting the next task hence it could be completed or half a task could be completed before starting the next one and can be completed in an overlapping time period.

   Parallelism is having multiple processes working simultaneously on multiple CPU's. This is where tasks are split into subtasks which can be processed in parallel on multiple CPU's at the exact same time. Multiple tasks are not needed as parts of tasks/multiple tasks can be run at the same time using the multiple CPU's when one core is assigned to each task/sub task.

   The main differences between concurrency or parallelism is that concurrency makes two or more actions happen at the same time but not at exactly the same instant whereas parallelism makes two or more actions happen simultaneously. Concurrency is done by a single central processing unit with the use of context switching whereas parallelism is done through multiple-single central processing units. Concurrency deals with a lot of tasks at the same time whereas Parallelism actually completes a lot of things at the same time.

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

   Gil is a Python process lock which prevents multi-threading from taking place hence only one thread is allowed to be executed at a time. It works by being a single lock on the interpreter which states for any Python bytecode to run you need to obtain the interpreter lock which prevents deadlocks. Therefore GIL helps to enhance the performance of programs that are run by single threads.

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

    DRY means "don't repeat yourself" to help limit the repetition of instructions, logic or code during the development of a process.

    KISS means "keep it simple stupid" to help make the code concise and clear to allow people to easily interpret it.

    BDUF means "big design up front" to ensure that everything is well thought through before implementing code.

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

    A garbage collector is used when the object is not being used and should be then released from memory by destroying it and freeing up memory for new objects. Python has an automated garbage collector and has two ways of removing the objects. The first is through reference counting where the reference count of the object is incremented when the object is referenced and vice vera. Therefore, if the reference count of the object is 0 then the memory for the object is removed and allocated elsewhere. The second method is through generational garbage collection which keeps track of all the objects in memory and divides the heap into three generations where the objects are placed and scanned in the youngest generation which are deallocated. However, if there are some objects remaining after multiple scans where each generation has a maximum limit of the number of objects it can hold and causes the collection process. If there are remaining objects they are then placed into the next older generation.

12. How is memory managed in Python?

    Memory is managed in Python by the interpreter and uses dynamic memory allocation, so you have no control over how the objects are managed. as it has a memory manager to manage the private heap which contains all the python objects and data structures. The memory manager contains object specific allocators to allocate memory for specific data objects and the raw memory allocated will work with the memory manager to make sure there is space ion the heap. The chunks of memory that are managed by the Python memory manager are called “blocks”. The stack memory is a stack frame which is created to store methods and variables where once they are returned the frame is destroyed and these objects will be collected by the garbage. Whereas the heap memory is used to store objects and instance variables. Different data types will be managed in a different way in the heap due to their difference in storage requirements.

13. What is a Python module?

    A file which contains title.py is a module as they refer to a file that contains Python code. Modules are used to break down larger programs into smaller files that are more organised and manageable. A module can contain function definitions and statements that can be executed and they can be imported into other modules.

14. What is docstring in Python?

    A docstring is a string literal that can be used to document a Python module, function, method or class as it normally appears after them. This allows people who are reading the document are able to understand what is going on without having to read through the code.

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

    Pickling is used to convert a python object into a byte stream, so it contains all the information necessary to reconstruct the object within another script that uses Python. Pickling uses two methods where the first is a dump where you dump the object and second is load which loads the object from another file.

    Pickling in Python:

    ```
    import pickle

    new_list = [12, "red", "green"]

    with open("data.pickle", "wb") as file_handle":
            pickle.dump(my_list, file_handle, pickle.HIGHEST_PROTOCOL)
    print("Pickling completed!")
    ```

    Pickling completed

    Unpickling is used to use to get back the original python objects from the stored pickle file. So it converts a byte stream back into the python object.

    Unpickling in Python:

    ```
    import pickle

    with open("data.pickle", "rb") as file_handle":
            retrieved_data = pickle.load(file_handle)
            print(retrieved_data)
    ```

    [12, "red", "green"]

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

    Pychecker is an opensource tool which helps to find bugs in Python from the source code and will give a warning about the style and how complex the bug is. This can be installed by pip package pip install Pychecker.

    Pylint is is an opensource tool that can be varied and has programs that help to control the warnings and errors. It helps to check for errors in the programme as well as the length of each line. It also checks the names of the variables and sees if it is compatible with the style of the project. This can be installed by pip install Pylint.

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

    Python uses pass by reference or pass by object reference where what is being reference is an object. This means the memory address of the variable is passed instead of coping the values of the variables and the code is executed on the value stored at the address. Therefore, a change in the variable will also affect the value of the variable outside the operation hence changes in the values of the variables will be permitted with calls to the function as a result it will modify the original value. When passing by reference if a change is made to a variable inside the function then as a result of this the change will also be seen outside the value as well.

    Example:

```
names = {'Maya': 17, 'Preet': 22, 'Zane': 28}

def age(names):
    new_names = {'Arooza':15, 'Chandni':21}
    names.update(new_names)
    print("Inside the function", names)
    return
age(names)
print("Outside the function:", names)
```

    Inside the function {'Maya': 17, 'Preet': 22, 'Zane': 28, 'Arooza':15, 'Chandni':21}
    Outside the function {'Maya': 17, 'Preet': 22, 'Zane': 28, 'Arooza':15, 'Chandni':21}

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

List comprehension uses one line of code when you want to create a new list from values that are from an existing list.

Without list comprehension:

```
pets = ["cat", "dog", "mouse", "rabbit", "hamster"]
result = [ ]

for i in pets:
  if "o" in i:
   result.append(i)

print(result)
```

[“dog”, “mouse”]

With list comprehension

```
pets = ["cat", "dog", "mouse", "rabbit", "hamster"]

result = [i for i in pets if "o" in i]

print(result)
```

[“dog”, “mouse”]

Dictionary comprehensions uses one line of code to allow you to create a dictionary.

Without Dictionary comprehension:

```
words = [“red”, “blue”, “yellow”, “purple”]
result = {}

for i in words:
 if len(i)>5:
   result[i] = len(i)

print(result)
```

{“yellow”: 6, “purple”:6}

With Dictionary comprehension:

```
words = [“red”, “blue”, “yellow”, “purple”]

result = {i:len(i) for i in words if len(i)>5}

print(result)
```

{“yellow”: 6, “purple”:6}

19. What is namespace in Python?

    A namespace is system that has a unique name for each object in python so they can be used without causing any conflict.

20. What is pass in Python?

    Pass is used as a placeholder where the pass statement is a null statement. Python does not ignore the pass but when it is executed there is no operation.

21. What is unit test in Python?

    A unit test is a smaller test to check if a single component is operating in the correct format therefore it will help you to detect what is not working in your application and help you to fix it faster.

22. In Python what is slicing?

    Slicing is a feature that allows you to access parts of sequences like strings, tuples and lists which can also be used to update or delete items of objects that are mutable.

23. What is a negative index in Python?

    Negative indexes start at -1 at the end of the element and can be used to help decide where you want to slice it. For example in an array the negative index will start from the end of it and each element to the left will have an index incrementing by 1.

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

    A ternary operator are conditional expressions that will evaluate the condition based on it being True or False. The ternary operatory allows you to test for the condition using only fewer lines of code and will therefore replace an if else statement that is used on multiple lines.

    Without a ternary operator:

```
if energetic_dog == True:
        print("You take the dog out for a walk.")
else:
        print("Me and dog will stay inside.")
```

With a ternary operator:

energetic_dog == True:
print("You take the dog out for a walk." if energetic_dog else "Me and dog will stay inside.")

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

    We use *args and **kwargs as an argument when we are do not know how many arguments should be passed into the function. Args is used to pass a variable that is non-keyword argument list where the operation of the list is performed whereas kwargs passes a variable number of keyword arguments dictionary to the function where dictionary operations are performed. Therefore, Args is used to pass a keyword argument into the function of a variable length argument.

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

    Range() and xrange() are used to tell you the number of times you should go through the for loop in python. Python 3 does not use xrange() as the range() behaves the same way but Python 2 does. Range() is known to return a range object that is of the type iterable whereas xrange() returns a generator object that is used only in looping to show numbers. In Python 2 range() creates a list in memory whereas xrange() gives a sequence object that has a lazy evaluation.

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

    Flask is a micro framework which is used for very few dependencies on external libraries. It can be used to create API's and web applications in python by providing tools, libraries and technologies so it is designed to be used quickly and with ease to help build more complicated applications.

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

    Clustered index uses the key values to sort the data rows in the table and in a database per table you will find only one clustered index so it can only be stored one way.

    With a non-clustered index is stored in a separate location to the data table where there is a second list that has pointers to the rows. You can have more than one non-clustered index, however on addition of each new index the time taken to write new records will also increase.

29. What is a 'deadlock' a relational database?

A deadlock takes place when two or more transactions are waiting for the lock to be released and each lock is held by the other lock, so all the processes are in the waiting state until they give up the locks.

30. What is a 'livelock' a relational database?

A livelock takes place when two or more transactions are constantly repeating the same interactions responding to the changes in the other processes without making progress. As the overlapping shared locks keep hindering one another this causes the task to be incomplete. Therefore they are not in the waiting state and as the states are constantly changing which is also known as a particular case of resource starvation.

2. Python string methods: describe each method and provide an example	**29 points**

METHOD	DESCRIPTION	EXAMPLE
capitalize()	A copy of the original string is returned and whilst converting the first character of the string to an uppercase letter whilst all the other characters in the string are lowercase letters.	`txt = "hello, my name is Vani."` `x = txt.capitalize()` `print(x)` **Solution** `"Hello, my name is Vani."`
casefold()	This method returns a string where all the characters are lower case. Although it is similar to the lower() method the casefold() method is more powerful. As a result it will convert more characters into lowercase and will find more matches when comparing two strings.	`string = "I AM LEARNING PYTHON."` `print("This is a lowercase string:", string.casefold())` **Solution** `This is a lowercase string: i am learning python.`

	.	
center()	This method will align the string in the centre with a specific number of characters and spaces that will be inputed by default to fill the character.	`txt = "doughnut"` `x = txt.center(20)` `print(x)` **Solution** **doughnut** **(the spaces will match the other side)**
count()	This method returns the number of times the value that has been specified is within the string.	`txt = "I love water. I find water so refreshing."` `x = txt.count("water")` `print(x)` **Solution** **2**
endswith()	If the string ends with the value specified then this function will return true.	`txt = "I love the lion king!"` `x = txt.endswith("!")` `print(x)` **Solution** **True**
find()	Similar to indexing he find() finds the index equivalent to the first character of the specified value. But it will return -1 if the method is not found whereas the index method would raise an exception if the value is not found.	`txt = "My house is beautiful."` `x = txt.find("house")` `print(x)` **Solution**

		3
format()	This method formats the values required and puts them into the strings placeholder which returns the formatted string. We use the curly brackets {} to define the placeholder.	txt = "Can you believe this costs only {price:.2f} pounds!" print(txt.format(price = 21)) **Solution** Can you believe this costs only 21 pounds!
index()	This method will return the position of the first time this value that is requested occurs.	colours = [‘red’, ‘blue’, ‘green’] x = colours.index("blue”) print(x) **Solution** 1
isalnum()	This method will only return True if all the characters are alphanumeric and there are no special characters.	txt = "gardenparty123” x = txt.isalnum() print(x) **Solution** true
isalpha()	This method will only return True if all the characters are present in the alphabet and there are no special characters or numbers.	txt = "Retainers" x = txt.isalpha() print(x) **Solution** true

isdigit()	This method will only return True if all the characters are digits including exponents such as squared.	`txt = "19786"` `x = txt.isdigit()` `print(x)` **Solution** **true**
islower()	This function will check if all the alphabet characters only are all lowercase. If they are then they will return True.	`txt = "the day is so sunny!"` `x = txt.islower()` `print(x)` **Solution** **true**
isnumeric()	This method will return True if the characters are all numeric. Whole number and fractional exponents count as numeric values whereas negative values and decimals do not and hence will return False.	`txt = "923678"` `x = txt.isnumeric()` `print(x)` **Solution** **true**
isspace()	If all the characters in a string are white spaces only then this function will return True.	`txt = "  "` `x = txt.isspace()` `print(x)` **Solution** **true**
istitle()	This function will return True if every word begins with an uppercase letter and the other letters are lowercase but symbols and numbers are ignored.	`txt = "Python Is The Best Language!"` `x = txt.istitle()` `print(x)`

		Solution true
isupper()	If all the characters are upper case then this method will return True however it only checks for alphabet characters.	`txt = "THIS IS SO MUCH FUN!"` `x = txt.isupper()` `print(x)` **Solution** true
join()	This method joins all the items into one string from the iterarable. Before the join a separator must be specified in a string even if it is an empty space.	`myTuple = ("Cat", "Dog", "Mouse")` `x = "!".join(myTuple)` `print(x)` **Solution** Cat!Dog!Mouse
lower()	This method returns a string where each character is lowercase but it does not include symbols or numbers.	`txt = "HOPE EVERYONE IS WELL"` `x = txt.lower()` `print(x)` **Solution** hope everyone is well
lstrip()	A string is returned where the whitespaces at the beginning or the left side of the string are removed.	`txt = "!!!!!books!!!!!"` `x = txt.lstrip()` `print(x)` **Solution** books!!!!!

replace()	This method replaces a targeted phrase with another sentence/phrase that is specified.	`txt = "I love chocolate"` `x = txt.replace("chocolate", "milk")` `print(x)` **Solution** **I love milk**
rsplit()	This method splits the string and starting from the right side as a result of this a list is produced. If no maximum value is specified then then what is returned will be the same as when using the split method.	`txt = "pink, blue, green"` `x = txt.rsplit(", ")` `print(x)` **Solution** **['pink', 'blue', 'green']**
rstrip()	A string is returned where the whitespaces at the right side or trailing of the string are removed. Spaces are a default trailing character.	`txt = "  orange  "` `x = txt.rstrip()` `print("my favourite fruit is", x, "it is the best")` **Solution** **my favourite fruit is orange it is the best**
split()	This method returns a list by splitting a string where a default separator is a whitespace and you can use any separator.	`txt = "my friends are lovely"` `x = txt.split()` `print(x)` **Solution** **["my", "friends", "are", "lovely"]**

splitlines()	This method returns a list by splitting the string where the splits are done at the line breaks.	`txt = "Wow thank you for the cake\nIt was delicious!"` `x = txt.splitlines()` `print(x)` **Solution** **["Wow thank you for the cake", "It was delicious!"]**
startswith()	This function will return True if the string starts with the value that is specified otherwise it will return False.	`txt = "We love your shoes."` `x = txt.startswith("We")` `print(x)` **Solution** **True**
strip()	This method removes any whitespaces at the beginning of the string and any trailing spaces at the end of the string where the default leading character is space.	`txt = "   cereal   "` `x = txt.strip()` `print("I love", x, "it is my favourite thing to eat.")` **Solution** **I love cereal it is my favourite thing to eat.**
swapcase()	This method returns a string which swaps the letters that are uppercase with those that are lowercase and vice versa.	`txt = "I loVE GINgeRbreAD"` `x = txt.swapcase()` `print(x)` **Solution** **i Love ginGErBREad**

title()	This returns a string where for each word the first character is uppercase. If the word has a number or special character then the letter after that will be uppercase.	txt = “Earth is the best planet” x = txt.title() print(x) **Solution** **Earth Is The Best Planet**
upper()	A string is returned where all of the characters are uppercase,	txt = "It is so nice to meet you" x = txt.upper() print(x) **Solution** **IT IS SO NICE TO SEE YOU**

3. Python list methods: describe each method and provide an example	**11 points**

Method	Description	Example
append()	This method adds or appends an element to the end of the list.	colours = ['red', 'blue', 'green'] colours.append("pink") print(colours) **Solution** **[‘red’, ‘blue’, ‘green’, ‘pink’]**
clear()	This method removes all the elements from the list and returns an empty list.	colours = ['red', 'blue',

		'green'] colours.clear() print(colours) **Solution** []
copy()	This method returns a copy of the list that is specified.	colours = ['red', 'blue', 'green'] x = colours.copy() print(colours) **Solution** ['red', 'blue', 'green']
count()	This method returns the number of times the value is specified in the string.	txt = "I love cake, cake is my favourite dessert" x = txt.count("cake") print(x) **Solution** 2
extend()	This method adds the requested list of elements of iterable at the end of the first list.	colours = ['red', 'blue', 'green'] clothes = ['shirt', 'socks', 'shorts'] colours.extend(clothes) print(colours)

		Solution ['red', 'blue', 'green', 'shirt', 'socks', 'shorts']
index()	This method returns the index position the very first time this value is specified.	`colours = ['red', 'blue', 'green']` `x = colours.index("blue")` `print(x)` **Solution** 1
insert()	This method adds the element specified at the position specified in the variable above.	`colours = ['red', 'blue', 'green']` `colours.insert(1, "pink")` `print(colours)` **Solution** ['red', 'pink', 'blue', 'green']
pop()	This removes an element at the position that is requested.	`colours = ['red', 'blue', 'green']` `print(colours)` **Solution** `colours.pop(1)` ['red', 'green']

remove()	This removes the first time the element specified occurs.	`colours = ['red', 'blue', 'green']` `colours.remove("blue")` `print(colours)` **Solution** ['red', 'green']
reverse()	This reverses the order the elements are sorted in.	`colours = ['red', 'blue', 'green']` `colours.reverse()` `print(colours)` **Solution** ['green', 'blue', 'red']
sort()	By default the list is sorted by ascending order.	`colours = ['red', 'blue', 'green']` `colours.sort()` `print(colours)` **Solution** ['blue', 'green', 'red']

4. Python tuple methods: **describe each method and provide an example**	**2 points**

Method	Description	Example
count()	This method returns the total number of times the value that is specified occurs in the tuple.	`newtuple = (2, 5, 9, 1, 2, 7, 3, 2)` `x = newtuple.count(2)` `print(x)` **Solution** 3
index()	This method will return the index number of the first time the specified value occurs in the tuple. If the method can not find the value then it will raise an exceotion.	`newtuple = (2, 5, 9, 1, 2, 7, 3, 2)` `x = newtuple.index(5)` `print(x)` **Solution** 1

5. Python dictionary methods: describe each method and provide an example	11 points

Method	Description	Example
clear()	This method will remove all the elements from the dictionary and will return a set of empty brackets.	`grades = {` `  "Bia": 88,` `  "Tim": 91,` `  "Maya": 64` `}` `grades.clear()` `print(grades)`

		Solution {}
copy()	This method will return a copy of the dictionary that is specified.	grades = { "Bia": 88, "Tim": 91, "Maya": 64 } x = grades.copy() print(x) **Solution** grades = { "Bia": 88, "Tim": 91, "Maya": 64 }
fromkeys()	This method will return a dictionary with the keys and values requested.	x = ('car1', 'car2', 'car3') y = 0 thisdict = dict.fromkeys(x, y) print(thisdict) **Solution** grades = { "car1": 0, "car2": 0, "car3”: 0 }

get()	This method will return the value of the item that is aligned with the key that is requested.	`grades = {` `  "Bia": 88,` `  "Tim": 91,` `  "Maya": 64` `}` `x =` `grades.get("Tim)` `print(x)` **Solution** `91`
items()	This method will return a view object which contains key value pairs of the dictionary as tuples in a list.	`grades = {` `  "Bia": 88,` `  "Tim": 91,` `  "Maya": 64` `}` `x =` `grades.items()` `print(x)` **Solution** `dict_items([('Bia', 88), ('Tim', 91), ('Maya', 64)])`
keys()	This method will return a view object which will have dictionary keys within a list.	`grades = {` `  "Bia": 88,` `  "Tim": 91,` `  "Maya": 64` `}` `x =` `grades.keys()` `print(x)` **Solution**

		dict_keys([‘Bia’, ‘Tim’, ‘Maya’])
pop()	This function removes a specific item from the dictionary. Where the value of the remaining dictionary after the item is removed is what is returned.	grades = { "Bia": 88, "Tim": 91, "Maya": 64 } grades.pop("tim") print(grades) **Solution** {"Bia": 88, "Maya”: 64}
popitem()	This function will remove the last item that is present in the dictionary and will return what is remaining.	grades = { "Bia": 88, "Tim": 91, "Maya": 64 } grades.popitem() print(grades) **Solution** {"Bia": 88, "Tim": 91}
setdefault()	The value of the key that is specified is returned. But if the key does not exist then insert the key and specific value then you can retrieve the value.	grades = { "Bia": 88, "Tim": 91, "Maya": 64 } x = grades.setdefault("Tim"

		`` print(x) ``` **Solution** ``` 91 ```
update()	This method inserts the items that are requested into the dictionary where the items themselves can be a dictionary or an object that contains key value pairs.	``` grades = { "Bia": 88, "Tim": 91, "Maya": 64 } grades.update({" Rhia": "52"}) print(grades) ``` **Solution** ``` grades = { "Bia": 88, "Tim": 91, "Maya”: 64, “Rhia”:52 } ```
values()	This method returns a view object which is a list containing the dictionary values.	``` grades = { "Bia": 88, "Tim": 91, "Maya": 64 } x = grades.values() print(x) ``` **Solution** ``` dict_values([88, 91, 64]) ```

6. **Python set methods:** **describe each method and provide an example**	**12 points**

Method	Description	Example
add()	This method is used to add an item to the set.	`cutleryset = {"fork", "spoon", "knife"}` `cutleryset.add("plate")` `print(cutleryset)` **Solution** {'spoon', 'plate', 'knife', 'fork'}
clear()	This method returns an empty set by removing all the elements in the set.	`cutleryset = {"fork", "spoon", "knife"}` `cutleryset.clear()` `print(cutleryset)` **Solution** set()
copy()	This returns a copy of the set.	`cutleryset = {"fork", "spoon", "knife"}`

		x = cutleryset.copy( ) print(x) **Solution** {‘knife’, ‘spoon’, ‘fork’}
difference()	This method returns only items that exist in the first set and not in both sets so it gives the difference between them both.	x = {"fork", "spoon", "knife"} y = {"fork", "plate", "bowl"} z = x.difference(y) print(z) **Solution** {‘spoon’, ‘knife’}
intersection()	This method returns items that exist in either two or more sets.	x = {"fork", "spoon", "knife"} y = {"fork", "plate", "bowl"} z = x.intersection(y) print(z) **Solution**

		{'fork'}
issubset()	This method will return True if the items in the set exists in the set that is specified else it will return False.	x = {"d", "e", "f"} y = {"f", "e", "d", "x", "y", "z"} z = x.issubset(y) print(z) **Solution** true
issuperset()	If all the items in the specified set exist in the original set then it will return True else it will return False. Here we are checking if one set is a superset of another set.	x = {"f", "e", "d", "x", "y", "z"} y = {"d", "e", "f"} z = x.issuperset(y) print(z) **Solution** true
pop()	This method will return a set that that removes the item specified in the set. If nothing is specified it will remove the last item by default.	cutleryset = {"fork", "spoon", "knife"} cutleryset.pop() print(cutleryset) **Solution** {'fork', 'spoon'}

remove()	This function will remove the element that is requested in the set. If the item specified does not exist then it will raise an error however the discard() does not raise an error.	cutleryset = {"fork", "spoon", "knife"} cutleryset.remove("spoon") print(cutleryset) **Solution** {'fork', 'knife'}
symmetric_difference()	This method will return items in a set that are not present in the both the sets.	X = {"fork", "spoon", "knife"} y = {"fork", "plate", "bowl"} z = x.symmetric_difference(y) print(z) **Solution** {'spoon', 'knife'}
union()	The union method will return the set of items from the initial set and the set of items from the requested set. If the item exists in both sets then it will only be present one time in the returned set. Union() created a different set z.	x = {"fork", "spoon", "knife"} y = {"fork", "plate", "bowl"} z = x.union(y) print(z) **Solution** {'spoon', 'plate', 'fork', 'bowl',

		‘knife’}
update()	This method updates the first set by adding items from another set or iterable to it. Similar to that of above if the item exists in both sets then it will only be present one time in the returned set. Update changes the current set and does not create a new one.	x = {"fork", "spoo n", "knife"} y = {"fork", "plate" , "bowl"} x.update(y) print(x) **Solution** {‘plate’, ‘spoon’, ‘bowl’, ‘knife’, ‘fork’}

7. **Python file methods:** **describe each method and provide an example**	**5 points**

Method	Description	Example
read()	This method will read the most size bytes from the file but if it reaches the end of the file before obtaining the size bytes then it will only read the available bytes. the default is set to -1 then the whole file will be returned.	f = open("examplefile .txt", "r") print(f.read())
readline()	This method will return a line from the file where the number of bytes can be specified from the line using the size parameter. By default the value is -1 and this means that the whole line will be returned.	f = open ("examplefile .txt", "r") print(f.readline( ))

readlines()	This method will return a list where each line that was present in the file is an item in the list. However if the total number of bytes are returned exceeds the number that is specified then no other lines will be returned. By default the value is -1 and this means that all of the lines will be returned.	`f = open("examplefile.txt", "r")` `print(f.readlines())`
write()	This method is used to create a file and write a specific text to the file and will be inserted depending on the file mode and the stream position when the file specified is not already present. "a" allows the text to be inserted without deleting any previous text any by default it will be inserted at the end of the file. "w" will empty the file before any new text gets inserted where the default value is 0.	`f = open("examplefile.txt", "w")` `f = open("examplefile.txt", "a")` `f.write("Nice to meet you!")` `f.close()` `# you can now read the file to see what is in it` `f = open("examplefile.txt", "r")` `print(f.read())`
writelines()	This method will take the items of the list and will write them to a file. "a" allows the text to be inserted without deleting any previous text any by default it will be inserted at the end of the file. "w" will empty the file before any new text gets inserted where the default value is 0.	`f = open("examplefile.txt", "a")` `f.writelines(["\nNice to meet you!")!", "\nHope to see you again."])` `f.close()` `# you can now read the file to see what is in it` `f = open("examplefi`

		le.txt", "r") print(f.read())