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 high-level programming language with dynamic semantics. The main features are:

- Easy to code: easy to learn the language as compared to other languages
- Free and Open Source:It means that Python source code is freely available to the public for download
- Object-Oriented Language: One of the key features of python is Object-Oriented programming
- High-Level Language: When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.
- Extensible feature: We can write us some Python code into C or C++ language
- Python is Portable language: Python code for windows can be run on other platforms such as Linux, Unix, and Mac
- Interpreted Language: Python code is executed line by line at a time, there is no need to compile python code.

● Dynamically Typed Language: That means the type for a variable is decided at run time not in advance, so we don't need to specify the type of variable.

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.

Reference:
https://www.guru99.com/python-2-vs-python-3.html#:~:text=KEY%20DIFFERENCE&text=Python%203%20default%20storing%20of,using%20it%20inside%20for-loop.

Python 3 supports modern techniques like AI, machine learning, and data science.

3. What is PEP 8?

Pep8 is a tool to check a Python code against some of the style conventions in PEP 8. The Pep8 statement is: *A style guide is about consistency. Consistency with this style guide is important. Consistency within a project is more important. Consistency within one module or function is most important.*

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

A program is a specific set of ordered operations for a computer to perform. In other words, it is a collection of instructions executed by a computer to perform a task from a specific command.

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

A process is a program that is running on your computer. This can be anything from a small background task, such as a spell-checker or system events handler to a full-blown application like Internet Explorer or Microsoft Word. The term "process" can also be used as a verb, which means to perform a series of operations on a set of data. For example, your computer's CPU processes information sent to it by

various programs. Reference: https://techterms.com/definition/process

6. In computing / computer science what is cache?

Cache is a high-speed access area that's a reserved section of main memory or an area on the storage device. The two main types of cache are memory cache and disk cache. With Internet browsers, cache is a temporary storage area where website data is stored. By caching this data, the web browser can improve performance by loading data from your disk, instead of the Internet, if it's ever needed again. A cache server is a computer or network device set up to store web pages that were accessed by users on a network. Any user trying to access a web page stored on the cache server is sent the stored version, instead of downloading the web page again. Cache servers help reduce network and Internet traffic congestion, and save the company on bandwidth costs.

Reference:https://www.computerhope.com/jargon/c/cache.htm)

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

With computer programming, 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. A program is capable of having multiple threads open at once and terminates or suspends them after the task is completed or the program is closed. A multithreading CPU is capable of executing multiple threads concurrently.

Threads are the virtual components or codes, which divides the physical core of a CPU into virtual multiple cores. A single CPU core can have up-to 2 threads per core. For example, if a CPU is dual core (i.e., 2 cores) it will have 4 threads. And if a CPU is Octal core (i.e., 8 core) it will have 16 threads and vice-versa.

Reference: https://www.computerhope.com/jargon/t/thread.htm; https://www.geeksforgeeks.org/what-are-threads-in-computer-processor-or-cpu/

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

Concurrency means executing multiple tasks at the same time but not necessarily simultaneously. There are two tasks executing concurrently, but those are run in a 1-core CPU, so the CPU will decide to run a task first and then the other task or run half a task and half another task, etc.

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. It enables single sequential CPUs to do lot of things "seemingly" simultaneously.

1. Concurrency is the task of running and managing the 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 parallelism is achieved through multiple central processing units(CPUs).
3. Concurrency can be done by using a single processing unit. While parallelism 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 parallelism improves the throughput and computational speed of the system.
5. Concurrency deals with a lot of things simultaneously. While parallelism does a lot of things simultaneously.
6. Concurrency is the non-deterministic control flow approach. While parallelism is a deterministic control flow approach.
7. In concurrency debugging is very hard. While in parallelism debugging is also hard but simpler than concurrency.

Reference:
https://www.geeksforgeeks.org/difference-between-concurrency-and-parallelism/

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

The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter. The impact of the GIL isn't visible to developers who execute single-threaded programs, but it can be a performance bottleneck in CPU-bound and multi-threaded code.

The problem was that this reference count variable needed protection from race conditions where two threads increase or decrease its value simultaneously. If this happens, it can cause either leaked memory that is never released or, even worse, incorrectly release the memory while a reference to that object still exists. This can cause crashes or other "weird" bugs in your Python programs.

This reference count variable can be kept safe by adding locks to all data structures that are shared across threads so that they are not modified inconsistently.

But adding a lock to each object or groups of objects means multiple locks will exist which can cause another problem—Deadlocks (deadlocks can only happen if there is more than one lock). Another side effect would be decreased performance caused by the repeated acquisition and release of locks.

Python uses reference counting for memory management. It means that objects created in Python have a reference count variable that keeps track of the number of references that point to the object. When this count reaches zero, the memory occupied by the object is released.

Multi-processing vs multi-threading: The most popular way is to use a multi-processing approach where you use multiple processes instead of threads. Each Python process gets its own Python interpreter and memory space so the GIL won't be a problem. Python has a multiprocessing module which lets us create processes.

Reference: https://realpython.com/python-gil/

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

DRY (Don't Repeat yourself): DRY usually refers to code duplication. If we write the same logic more than once, we should "DRY up our code."

KISS (Keep it simple Stupid): The simpler and not over complicating a solution the better. Writing efficient, efficient, and simple code is great.

BDUF (Big Design Up Front): Planning and understanding the flaws and the process helps to build a better solution and architecture with less errors.

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

A garbage collector in Python is an automated algorithm that deallocates objects no longer needed, freeing up memory space. It works through generational garbage collection or reference counting. When the reference count of an object reaches 0, the reference counting garbage collection algorithm cleans up the object immediately. If you have a cycle, the reference count doesn't reach zero, you wait for the generational garbage collection algorithm to run and clean the object.

Reference: https://stackify.com/python-garbage-collection/
https://towardsdatascience.com/memory-management-and-garbage-collection-in-python-c1cb51d1612c

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. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.

13. What is a Python module?

A module is a file containing Python definitions and statements. A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module name is encountered in an import statement.

Reference: https://docs.python.org/3/tutorial/modules.html

14. What is docstring in Python?

A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition.

Docstring conventions are described within PEP 257. Their purpose is to provide your users with a brief overview of the object. They should be kept concise enough to be easy to maintain but still be elaborate enough for new users to understand their purpose and how to use the documented object.

In all cases, the docstrings should use the triple-double quote (""") string format. This should be done whether the docstring is multi-lined or not. At a bare minimum, a docstring should be a quick summary of whatever is it you're describing and should be contained within a single line

https://www.python.org/dev/peps/pep-0257/
https://realpython.com/documenting-python-code/

15. What is pickling and unpickling in Python? Example usage.
The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.

Pickling: It is a process where a Python object hierarchy is converted into a byte stream.

```
import pickle
emp = {1:"A",2:"B",3:"C",4:"D",5:"E"}
pickling_on = open("Emp.pickle","wb")
pickle.dump(emp, pickling_on)
pickling_on.close()
```

Unpickling: It is the inverse of the Pickling process where a byte stream is converted into an object hierarchy.

```
pickle_off = open("Emp.pickle","rb")
emp = pickle.load(pickle_off)
print(emp)
```

https://www.geeksforgeeks.org/pickle-python-object-serialization/

16. What are the tools that help to find bugs or perform static analysis?
PyChecker is a tool for finding bugs in python source code. It finds problems that are typically caught by a compiler for less dynamic languages, like C and C++. Pylint is a tool that checks for errors in Python code, tries to enforce a coding standard and looks for code smells. It can also look for certain type errors, it can recommend suggestions about how particular blocks can be refactored and can offer you details about the code's complexity.

17. How are arguments passed in Python by value or by reference? Give an example.
All parameters (arguments) in the Python language are passed by reference.

```
student={'Archana':28,'krishna':25,'Ramesh':32,'vineeth':25}
def test(student):
  new={'alok':30,'Nevadan':28}
  student.update(new)
  print("Inside the function",student)
  return
test(student)
print("outside the function:",student)
```

18. What are Dictionary and List comprehensions in Python? Provide examples.
Lists are just like the arrays. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

```
# Creating a List with
# the use of multiple values
List = ["Geeks", "For", "Geeks"]
print("List containing multiple values: ")
print(List[0])
print(List[2])

# Creating a Multi-Dimensional List
# (By Nesting a list inside a List)
List = [['Geeks', 'For'] , ['Geeks']]
print("\nMulti-Dimensional List: ")
print(List)
```

Dictionary in Python on the other hand is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.

```
# Creating a Dictionary
# with Integer Keys
Dict = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print("Dictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary
# with Mixed keys
Dict = {'Name': 'Geeks', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)
```

https://www.geeksforgeeks.org/difference-between-list-and-dictionary-in-python/

19. What is namespace in Python?

A namespace is a mapping from names to objects. Most namespaces are currently implemented as Python dictionaries. Examples of namespaces are: the set of built-in names (containing functions such as abs(), and built-in exception names); the global names in a module; and the local names in a function invocation. In a sense the set of attributes of an object also form a namespace. The important thing to know about namespaces is that there is absolutely no relation between names in different namespaces; for instance, two different modules may both define a function maximize without confusion — users of the modules must prefix it with the module name.

https://realpython.com/python-namespaces-scope/

https://medium.com/swlh/mastering-python-namespaces-and-scopes-7eba67aa3094

20.What is pass in Python?

The pass statement is used as 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.
Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

https://www.w3schools.com/python/ref_keyword_pass.asp

21. What is unit test in Python?
Unit testing is a software testing method by which individual units of source code are put under various tests to determine whether they are fit for use (Source). It determines and ascertains the quality of your code. The unit test supports test automation, sharing of setup and shutdown code for tests, aggregation of tests into collections, and independence of the tests from the reporting framework.

https://docs.python.org/3/library/unittest.html

22. In Python what is slicing?
Slicing in Python allows you to cut an object. You can use the function slice to specify which part of the sequence you want to retrieve. Where to start, where to end and the step.

23. What is a negative index in Python?
Negative indexes are used to count from the end of the string. In the example: b = "Hello, World!"
print(b[-5:-2])
The output is: orl
Same output without negative index would be: b = "Hello, World!"
print(b[8:11])

24.How can the ternary operators be used in python? Give an example.
Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false.
[on_true] if [expression] else [on_false]
Example:
age = 48
discount = True if age >= 65 else False
print(discount)
False

25. What does this mean: *args, **kwargs? And why would we use it?
The special syntax *args in function definitions in python are used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list. The special syntax **kwargs in function definitions in python are used to pass a keyworded, variable-length argument list. *args and **kwargs allow you to pass an unspecified number of arguments to a function, so when writing the function definition, you do not need to know how many arguments will be passed to your function.

26. How are range and xrange different from one another?
In Python 3, there is no xrange , but the range function behaves like xrange in Python

2. range() – This returns a range object (a type of iterable).
xrange() – This function returns the generator object that can be used to display numbers only by looping. Only a particular range is displayed on demand and hence called “lazy evaluation“.

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

Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.

https://pymbook.readthedocs.io/en/latest/flask.html

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

A clustered index sorts and stores the full records in the actual table based on their key values (the indexed column). There is only one clustered index per table. It is usually the primary key. In MySQL, the server automatically creates a clustered index from the primary key. A table with a clustered index is called a clustered table. A non-clustering index is defined in the non-ordering field of the table. This type of indexing method helps you to improve the performance of queries that use keys which are not assigned as a primary key. A non-clustered index allows you to add a unique key for a table.

29. What is a ‘deadlock’ in a relational 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. Transaction B holds locks on those very rows in the Orders table but needs to update the rows in the Accounts table held by Transaction A. Transaction A cannot complete its transaction because of the lock on Orders. Transaction B cannot complete its transaction because of the lock on Accounts. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

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 with 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: describe each method and provide an example	29 points

METHOD	DESCRIPTION	EXAMPLE

capitalize()	Return a copy of the string with its first character capitalized and the rest lowercase.	`name = str('MAYUmI').capitalize()`
		`print(name)`
		`#Mayumi`

casefold()	Casefolded strings may be used for caseless matching.
	
	Casefolding is similar to lowercasing but more aggressive because it is intended to remove all case distinctions in a string. For example, the German lowercase letter 'ß' is equivalent to "ss". Since it is already lowercase, lower() would do nothing to 'ß'; casefold() converts it to "ss".
		`motivational = ('ALL study AND eFfort will pay off')`
		`motivational_casefold = motivational.casefold()`
		`print(motivational_casefold)`
		`#all study and effort will pay off`

center()	Return centered in a string of length width.	`header = str('Availability').center(50)`
		`print(header)`
		`#`
		`Availability`

count()	Return the number of non-overlapping occurrences of substring sub in the range [start, end].	`instructions = ('Include 3 liters of water and 2 liters of concentrate orange juice')`
		`liters = instructions.count('liters')`
		`print(liters)`
		`#2`

endswith()	Return True if the string ends with the specified suffix, otherwise return False.	`name = ('My name is Mayumi.')`
		`sentence = name.endswith(".")`
		`print(sentence)`
		`#True`

find()	Return the lowest index in the string where substring sub is found within the slice	`name = ('My name is Mayumi.')`

	s[start:end].	`sentence = name.find("is")`
		`print(sentence)`
		`#8`

format()	Perform a string formatting operation. The string on which this method is called can contain literal text or replacement fields delimited by braces {}.	`name = ('Disney')`
		`sentence = ('My name is {}').format(name)`
		`print(sentence)`
		`#My name is Disney`

index()	Return the lowest index in the string where substring sub is found within the slice s[start:end], but raise ValueError when the substring is not found.	`name = ('My name is Mayumi.')`
		`sentence = name.index("name")`
		`print(sentence)`
		`#3`

isalnum()	Return True if all characters in the string are alphanumeric and there is at least one character, False otherwise.	`instructions = ('Include 3 liters of water and 2 liters of concentrate orange juice')`
		`liters = instructions.isalnum()`
		`print(liters)`
		`#False`

isalpha()	Return True if all characters in the string are alphabetic and there is at least one character, False otherwise. Alphabetic characters are those characters defined in the Unicode character database as “Letter”, i.e., those with general category property being one of “Lm”, “Lt”, “Lu”, “Ll”, or “Lo”.	`price = 'Price'.isalpha()`
		`print(price)`
		`#True`

isdigit()	The isdigit() method returns True if all the characters are digits, otherwise False. Exponents, like ^{2}, are also considered to be a digit.	`price = '50.60'`
		`price_value = price.isdigit()`
		`print(price_value)`
		`#False`

islower()	Return True if all cased characters in the string are lowercase and there is at least one cased character, False otherwise.	`price = 'price is 50.60'.islower()`
		`print(price)`
		`#True`

isnumeric()	Return True if all characters in the string are numeric characters, and there is at least one character, False otherwise. Numeric	`price = '5060²'` `price_value = price.isnumeric()`

	characters include digit characters, and all characters that have the Unicode numeric value property, e.g. U+2155, VULGAR FRACTION ONE FIFTH.	`print(price_value)` `#True`

isspace()	Return True if there are only whitespace characters in the string and there is at least one character, False otherwise.	`name = ('  ')`
		`sentence = name.isspace()`
		`print(sentence)`
		`#True`

istitle()	Return True if the string is a titlecased string and there is at least one character, for example uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.	`name = ('My Name Is Mayumi.')`
		`sentence = name.istitle()`
		`print(sentence)`
		`#True`

isupper()	Return True if all cased characters in the string are uppercase and there is at least one cased character, False otherwise.	`print('BANANA'.isupper())`
		`# True`
		`print('banana'.isupper())`
		`# False`
		`print('baNana'.isupper())`
		`# False`
		`print('  '.isupper())`
		`# False`

join()	Return a string which is the concatenation of the strings in iterable. A TypeError will be raised if there are any non-string values in iterable, including bytes objects. The separator between elements is the string providing this method.	`classroom = ("Mary","Bella","Josh")`
		`classroom_csv = ", ".join(classroom)`
		`print(classroom_csv)`
		`#Mary, Bella, Josh`

lower()	Return a copy of the string with all the cased characters converted to lowercase.	`name = str('MAYUmI').lower()`
		`print(name)`
		`# mayumi`

lstrip()	Return a copy of the string with leading characters removed. The chars argument is a string specifying the set of characters to be removed. If omitted or None, the chars argument defaults to removing whitespace.	`ingredients = '     3 liters of water'`
		`instructions = ingredients.lstrip()`
		`print("Include", instructions, "in the bowl.")`
		`#Include 3 liters of water in the bowl.`

replace()	Return a copy of the string with all occurrences of substring old replaced by	`ingredients = '3 liters of water'`

	new. If the optional argument count is given, only the first count occurrences are replaced.	`instructions = ingredients.replace("water","orange juice")`
		`print("Include", instructions, "in the bowl.")`
		`#Include 3 liters of orange juice in the bowl.`

rsplit()	The rsplit() method splits a string into a list, starting from the right.	`classroom = ("John; Mary; Bella; Josh")`
		`classroom_list = classroom.rsplit(';')`
		`print(classroom_list)`
		`#['John', 'Mary', 'Bella', ' Josh']`

rstrip()	The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.	`ingredients = '3 liters of water,.,.,.,'`
		`instructions = ingredients.rstrip(',.')`
		`print("Include", instructions, "in the bowl.")`
		`#Include 3 liters of water in the bowl.`

split()	The split() method splits a string into a list.
	
	You can specify the separator, default separator is any whitespace.
		`classroom = ("Mary Bella Josh")`
		`classroom_list = classroom.split()`
		`print(classroom_list)`
		`#['Mary', 'Bella', 'Josh']`

splitlines()	Return a list of the lines in the string, breaking at line boundaries.	`instructions = ('Include 3 liters of water \n 2 liters of concentrate orange juice')`
		`instructions_lines = instructions.splitlines()`
		`print(instructions_lines)`
		`#['Include 3 liters of water ', ' 2 liters of concentrate orange juice']`

startswith()	Return True if string starts with the prefix, otherwise return False.	`name = ('My Name Is Mayumi.')` `sentence =`

		`name.startswith('Name')` `#False`

strip()	The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)	`ingredients = '  3 liters of water  '`
		`instructions = ingredients.strip()`
		`print("Include", instructions, "in the bowl.")`
		`#Include 3 liters of water in the bowl.`

swapcase()	Return a copy of the string with uppercase characters converted to lowercase and vice versa.	`name = str('My Name IS MAYumi').swapcase()`
		`print(name)`
		`#mY nAME is mayUMI`

title()	Return a titlecased version of the string where words start with an uppercase character and the remaining characters are lowercase.	`name = str('My Name IS MAYumi').title()`
		`print(name)`
		`#My Name Is Mayumi`

upper()	Return a copy of the string with all the cased characters converted to uppercase.	`name = str('My Name IS MAYumi').upper()`
		`print(name)`
		`#MY NAME IS MAYUMI`

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

Method	Description	Example

append()	Add an item to the end of the list.	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']`
		`fruits.append('grape')`
		`print(fruits)`
		`#['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange',`

		'grape']

clear()	Remove all items from the list.	`fruits = ['orange', 'apple', 'pear']`
		`fruits.clear()`
		`print(fruits)`
		`# []`

copy()	Return a shallow copy of the list.	`fruits = ["apple", "banana", "cherry"]`
		`print(fruits.copy())`
		`#['apple', 'banana', 'cherry']`

count()	Return the number of times x appears in the list.	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']`
		`print(fruits.count('apple'))`
		`#2`

extend()	Extend the list by appending all the items from the iterable.	`fruits = ['orange', 'kiwi', 'apple', 'banana']`
		`berries = ['strawberry','blueberry','grape']`
		`fruits.extend(berries)`
		`print(fruits)`
		`#['orange', 'kiwi', 'apple', 'banana', 'strawberry', 'blueberry', 'grape']`

index()	Return zero-based index in the list of the first item whose value is equal to x. Raises a ValueError if there is no such item.	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']`
		`print(fruits.index('banana',4))`
		`#6`

insert( )	Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert(0, x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x).	`berries = ['strawberry','blueberry','grape'] berries.insert(0,"cherry") print(berries) #['cherry', 'strawberry', 'blueberry', 'grape']`
pop( )	Remove the item at the given position in the list, and return it. If no index is specified, a.pop( ) removes and returns the last item in the list.	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana'] fruits.pop() print(fruits) #['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple']`
remove( )	Remove the first item from the list whose value is equal to x. It raises a ValueError if there is no such item.	`berries = ['strawberry','blueberry','grape'] berries.remove('strawberry') print(berries) #['blueberry', 'grape']`
reverse( )	Reverse the elements of the list in place.	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana'] fruits.reverse() print(fruits) #['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']`
sort( )	Sort the items of the list in place (the arguments can be used for sort customization, see sorted( ) for their explanation).	`fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana', ‘grape’] fruits.sort() print(fruits) #['apple', 'apple',`

		'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']

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

Method	Description	Example

count()	The count() method returns the number of times a specified value appears in the tuple.	`tuple = (1, 7, 5, 4, 8, 5)`
		`x = tuple.count(5)`
		`print(x)`
		`#2`

index()	The index() method finds the first occurrence of the specified value.
	The index() method raises an exception if the value is not found.
		`indextuple = (1, 7, 7, 5, 4, 6, 8)`
		`x = indextuple.index(8)`
		`print(x)`
		`#6`

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

Method	Description	Example

clear()	Remove all items from the dictionary.	`playlist = {`
		`"artist": "Jorja Smith",`
		`"song": "Be Honest",`
		`"duration": "3:27"`
		`}`
		`playlist.clear()`
		`print(playlist)`
		`#{}`

copy()	Return a shallow copy of the dictionary.	`playlist = {` `"artist": "Jorja Smith",`

		"song": "Be Honest", "duration": "3:27" } playlist.copy() print(playlist) #{'artist': 'Jorja Smith', 'song': 'Be Honest', 'duration': '3:27'}
fromkeys()	Create a new dictionary with keys from iterable and values set to value.	playlist = ("artist","song","duration") details = 0 playlist_dict = dict.fromkeys(playlist,details) print(playlist_dict) #{'artist': 0, 'song': 0, 'duration': 0}

get()	Return the value for key if key is in the dictionary, else default. If default is not given, it defaults to None, so that this method never raises a KeyError.	playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		song = playlist.get("song")
		print(song)
		#Be Honest

items()	Return a new view of the dictionary's items ((key, value) pairs).	playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		items = playlist.items()
		print(items)
		#dict_items([('artist', 'Jorja Smith'), ('song', 'Be Honest'), ('duration',

		'3:27')])

keys()	Return a new view of the dictionary's keys.	playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		keys = playlist.keys()
		print(keys)
		#dict_keys(['artist', 'song', 'duration'])

pop()	If key is in the dictionary, remove it and return its value, else return default. If default is not given and key is not in the dictionary, a KeyError is raised.	playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		which_artist = playlist.pop("song")
		print(which_artist)
		print()
		
		playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		playlist.pop("song")
		print(playlist)
		#Be Honest
		
		{'artist': 'Jorja Smith', 'duration': '3:27'}

popitem()	Remove and return a (key, value) pair from the dictionary. popitem() is useful to destructively iterate over a dictionary, as often used in set algorithms. If the dictionary is empty, calling popitem() raises a KeyError.	playlist = { "artist": "Jorja Smith",

		"song": "Be Honest", "duration": "3:27" } playlist.popitem() print(playlist) #{'artist': 'Jorja Smith', 'song': 'Be Honest'}

setdefault()	The setdefault() method returns the value of the item with the specified key. If the key does not exist, insert the key, with the specified value.	playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		artist = playlist.setdefault("artist")
		print(artist)
		#Jorja Smith

update()	Update the dictionary with the key/value pairs from other, overwriting existing keys. Return None.
	update() accepts either another dictionary object or an iterable of key/value pairs (as tuples or other iterables of length two). If keyword arguments are specified, the dictionary is then updated with those key/value pairs: d.update(red=1, blue=2).
		playlist = {
		"artist": "Jorja Smith",
		"song": "Be Honest",
		"duration": "3:27"
		}
		feat = playlist.update({"feat":"Burna Boy"})
		print(playlist)
		#{'artist': 'Jorja Smith', 'song': 'Be Honest', 'duration': '3:27', 'feat': 'Burna Boy'}

values()	Return a new view of the dictionary's values. An equality comparison between one dict.values() view and another will always return False. This also applies when comparing dict.values() to itself:	playlist = { "artist": "Jorja Smith", "song": "Be Honest", "duration": "3:27"

		} data = playlist.values() print(data) #dict_values(['Jorja Smith', 'Be Honest', '3:27'])

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

Method	Description	Example
add()	Once a set is created, you cannot change its items, but you can add new items. To add one item to a set use the add() method.	flowers_set = {'dandelion', 'rose', 'daisy'} flowers_set.add('lilies') print(flowers_set) # {'rose', 'daisy', 'lilies', 'dandelion'}
clear()	Removes all the elements from the set	flowers_set = {'dandelion', 'rose', 'daisy'} flowers_set.clear() #set()
copy()	Returns a copy of the set	flowers_set = {'dandelion', 'rose', 'daisy', ‘lilies’} flowers_set.copy() print(flowers_set) #{'daisy', 'lilies', 'dandelion', 'rose'}
difference()	Returns a set containing the difference between two or more sets	flower_set = {'lilies', 'rose', 'daisy'} herbs_set = {'mint','rose','basil'} distinct = herbs_set.differenc

		e(flowers_set) print(distinct) #{'mint', 'basil'}
intersection()	The intersection() method returns a set that contains the similarity between two or more sets. Meaning: The returned set contains only items that exist in both sets, or in all sets if the comparison is done with more than two sets.	flower_set = {'lilies', 'rose', 'daisy'} herbs_set = {'mint','rose','basil'} similar = herbs_set.intersection(flowers_set) print(similar) #{'rose'}
issubset()	Returns whether another set contains this set or not	flower_set = {'lilies', 'rose', 'daisy'} herbs_set = {'mint','rose','basil','lilies','daisy'} plants_set = flowers_set.issubset(herbs_set) print(plants_set) #True
issuperset()	Returns whether this set contains another set or not	flower_set = {'lilies', 'rose', 'daisy'} herbs_set = {'mint','rose','basil','lilies','daisy'} plants_set = flowers_set.issuperset(herbs_set) print(plants_set) #False
pop()	Removes a random element from the set	herbs_set = {'mint','rose','basil','lilies','daisy'} herbs_set.pop() print(herbs_set) #{'daisy', 'basil', 'mint', 'lilies'}

remove()	Removes the specified element	`flowers_set = {'dandelion', 'rose', 'daisy','lilies'}`
		`flowers_set.remove('rose')`
		`print(flowers_set)`
		`# {'dandelion', 'daisy', 'lilies'}`

symmetric_difference()	Returns a set with the symmetric differences of two sets	`flower_set = {'lilies', 'rose', 'daisy'}`
		`herbs_set = {'mint','rosemary', 'basil','rose'}`
		`plants_set = flowers_set.symmetric_difference(herbs_set)`
		`print(plants_set)`
		`#{'rosemary', 'mint', 'basil', 'lilies', 'daisy'}`

union()	Return a set containing the union of sets	`flower_set = {'lilies', 'rose', 'daisy'}`
		`herbs_set = {'mint','rosemary', 'basil','rose'}`
		`plants_set = flowers_set.union(herbs_set)`
		`print(plants_set)`
		`#{'daisy', 'mint', 'rose', 'basil', 'lilies', 'rosemary'}`

update()	To add items from another set into the current set, use the update() method.	`print('Example update')`
		`flower_set = {'lilies', 'rose', 'daisy'}`
		`herbs_set = {'mint','rosemary', 'basil','rose'}`
		`flowers_set.update(herbs_set)`

		`print(flowers_set)` `#{'lilies', 'basil', 'mint', 'rosemary', 'daisy', 'rose'}`

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

Method	Description	Example

read()	By default the read() method returns the whole text, but you can also specify how many characters you want to return.	`file = open("C:\\Users\\mayum\\Documents\\Trainings\\NANODEGREE\\1. Python lessons 1-6\\Session 5 (21-06-21)\\5.3_example_three.txt", 'r')`
		`#appending something to a file`
		`content = file.read()`
		`print(content)`
		`#content printed`

readline()	You can return one line by using the readline() method	`file = open("C:\\Users\\mayum\\Documents\\Trainings\\NANODEGREE\\1. Python lessons 1-6\\Session 5 (21-06-21)\\5.3_example_one.txt", 'r')`
		`content = file.readline()`
		`print(content)`

readlines()	Returns a list containing lines from the file.	`file = open("C:\\Users\\mayum\\Documents\\Trainings\\NANODEGREE\\1. Python lessons 1-6\\Session 5 (21-06-21)\\5.3_example_one.txt", 'r')` `content = file.readlines()`

		`print(content)` `#content printed in lines`

write()	Add content to the file.	`file = open("C:\\Users\\mayum\\Documents\\Trainings\\NANODEGREE\\1. Python lessons 1-6\\Session 5 (21-06-21)\\5.3_example_three.txt", 'w')`
		`file.write("Frankly, my dear")`
		`file.close()`
		`#Frankly, my dear added to file`

writelines()	Add content to the file in lines.	`file = open("C:\\Users\\mayum\\Documents\\Trainings\\NANODEGREE\\1. Python lessons 1-6\\Session 5 (21-06-21)\\5.3_example_three.txt", 'a')`
		`file.writelines("\n"+to_do+";")`
		`file.close()`
		`#append to_do to the file`