THEORY QUESTIONS ASSIGNMENT

Python based theory

Name: Shayli Patel

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 multi-purpose, high -level programming language.

   Its main features are that it is easy to code, it is free and open sourced for public use, it is portable so can be run on different platforms and it is an interpreted language.

2. Discuss the difference between Python 2 and Python 3

   Division operator - in Python 2, instead of the output being a float, it will be rounded to integer. In Python 3, the output will be a float.

   For example, = -7 / 5

   in Python 2 Output ---> 1

   in Python 3 Output ---> -1.4

   Print - print keyword in Python 2 is replaced by the print() function in Python 3

   Python 2 - print 'Hello World'

   Python 3 - print ('Hello World')

   Unicode - In Python 2, an implicit string type is ASCII. But in Python 3.x

implicit string type is Unicode.
xrange - xrange of Python 2 doesn't exist in Python 3.
Error Handling - minor change in error handling in both versions. In python 3.x, 'as' keyword is required.

3. What is PEP 8?
   PEP 8 (Python Enhancement Proposal 8) is a document that provides guidelines and best practices on how to write Python code. Its primary focus is to improve the readability and consistency of Python code.

4. In computing / computer science what is a program?
   A program is a set of instructions that are run by a computer.

5. In computing / computer science what is a process?
   A process is a program in execution.

6. In computing / computer science what is cache?
   Cache is a small amount of memory which is a part of the CPU. Cache is used to temporarily hold instructions and data that the CPU is likely to reuse.

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

   A thread is a single sequential flow of execution that can execute in parallel with other threads that are part of the same root process. Multithreading is a CPU (central processing unit) feature that allows two or more instruction threads to execute independently while sharing the same process resources.

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. Concurrency is used for multi-threading to alternate between different programmes. This back and forth between the different programmes takes longer however it uses less computation power.
   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 to run small chunks at the same time. A benefit of using parallelism is that it takes a shorter time however this uses more computational power.

9. What is GIL in Python and how does it work?
   Global Interpreter Lock is a lock that allows only one thread to hold the control of the Python interpreter and can be in a state of execution at any point in time. 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.

10. What do these software development principles mean:
    DRY - Don't Repeat Yourself
    The DRY principle states that these small pieces of knowledge may only occur exactly once in your entire system. Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.

    KISS - Keep It Simple, Stupid - a principle that one should try to keep a piece of software simple so that it easy to work on it later.
    BDUF - Big Design Up Front - a software development approach (often associated with the waterfall model) in which the program's design is to be completed and perfected before that program's implementation is started.

11. What is a Garbage Collector in Python and how does it work?
    The garbage collector is keeping track of all objects in memory. A new object starts its life in the first generation of the garbage collector. If Python executes a garbage collection process on a generation and an object survives, it moves up into a second, older generation.

12. How is memory managed in Python?
    The Python memory manager manages Python's memory allocations. Python uses a portion of the memory for internal use and non-object memory. The other portion is dedicated to object storage. The Python memory manager manages the Python heaps, which contain all Python objects and data structures, on demand. The Python memory manager has object-specific allocators to allocate memory for specific objects such as integers, strings. Raw memory allocator interacts with the memory manager of the operating system to ensure that there's space on the private heap.
    The Python memory manager manages chunks of memory called blocks. A collection of blocks of the same size makes up the Pool. Pools are created on Arenas, chunks of 256kB memory allocated on heap=64 pools. If the objects get destroyed, the memory manager fills this space with a new object of the same size.

13. What is a Python module?
    It is code that someone else has written that you can reuse in our programs. There are two types: built-in and user-defined

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. Docstrings are accessible from the doc attribute (__doc__) for any of the Python objects and also with the built-in help() function. An object's docstring is defined by including a string constant as the first statement in the object's definition.

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

    Python pickle module is used for serializing and de-serializing a Python object structure.

    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 is converted back into an object hierarchy.

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.

    Pylint is highly configurable and it acts like special programs to control warnings and errors, it is an extensive configuration file Pylint is also an opensource tool for static code analysis it looks for programming errors and is used for coding standard. It checks the length of each programming line. It checks the variable names according to the project style. It can also be used as a standalone program, it also integrates with python IDEs.

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. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

    Example:

```
students = { ‘Sam’ : 26 , ‘Niall’: 21 , ‘Scott’ : 24}
def test (students):
        new_students = {‘Dan’ : 23 , ‘Pete’ : 27}
        students.update(new_students)
        print(‘Inside the function : {}’.format( students))
        return
test(students)
print(‘Outside the function : {}’.format( students))
```

    Output

```
Inside the function : ‘Sam’ : 26 , ‘Niall’: 21 , ‘Scott’ : 24, ‘Dan’ : 23 , ‘Pete’ : 27
Outside the function : ‘Sam’ : 26 , ‘Niall’: 21 , ‘Scott’ : 24, ‘Dan’ : 23 , ‘Pete’ : 27
```

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

A list is an ordered collection of values and are written inside square brackets ([]) and separated by commas.

lottery_numbers = [4, 17, 28, 23, 49, 38]

A dictionary stores a collection of labelled items and are written in curly brackets ({}). Each item has a key and a value.

favourite_colour = { 'Polly': 'pink' , 'Nina' : 'blue', 'Sabrina' : 'purple'}

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.

There are four types of namespaces with differing lifetimes:

- **Built-In** - contains the names of all of Python's built-in objects and are available at all times when Python is running
- **Global** - contains any names defined at the level of the main program. Python creates the global namespace when the main program body starts, and it remains in existence until the interpreter terminates.
- **Enclosing** - remains in existence until its respective function terminates. This namespace encloses a local function.
- **Local** - remains in existence until its respective function. terminates.This namespace is local to the function and remains in existence until the function terminates.

20. What is pass in Python?

Pass is a null statement. The interpreter doesn't ignore a pass statement, but nothing happens and the statement results into no operation.

To avoid a compilation of errors, the pass statement can be used when you don't write the implementation of a function but want to implement it in the future.

21. What is unit test in Python?

A unit test is a scripted code level test designed in Python to verify a small "unit" of functionality.

22. In Python what is slicing?

Slicing creates a new substring from the source string and the original string remains unchanged.Each character can be accessed using their index number.

The slice() function returns a slice object. A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end and also specify the step. Syntax is slice(start, end, step).

The index number starts from 0 which denotes the first character of a string.

23. What is a negative index in Python?
    Negative indexing uses negative indexes to start the slice from the end of the string.

24. How can the ternary operators be used in python? Give an example.
    The ternary operator is a way of writing conditional statements and can be thought of as a simplified, one-line version of the if-else statement to test a condition.
    This Python operator consists of three operands:
    Condition: A boolean expression that evaluates to either *true* or *false*.
    true_val: A value to be assigned if the expression is evaluated to *true*.
    false_val: A value to be assigned if the expression is evaluated to *false*.

    Example: Check if integer is even.
    to_check = 8
    msg = 'Even' if to_check % 2 == 0 else 'Odd'
    print(msg)

25. What does this mean: *args, **kwargs? And why would we use it?
    *args and **kwargs allow you to pass multiple arguments or keyword arguments to a function.
    *args allows you to pass a varying number of positional arguments.
    Must use the unpacking operator (*).
    **kwargs works just like *args, but instead of accepting positional arguments it accepts keyword (or named) arguments.
    Must use the unpacking operator (**)

26. How are range and xrange different from one another?
    range() and xrange() are two functions that could be used to iterate a certain number of times in for loops in Python. 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.

27. What is Flask and what can we use it for?
    Flask is a micro or lightweight Python web framework, which is a third-party Python library used for developing web applications. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.
    it can be used for web applications,

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

A clustered index will cause our data rows to be physically stored in a certain order and sorted by the indexed column. Usually the clustered index is the primary key.

A non-clustered index does not physically sort out the data.It creates a map/ pointers to our actual raw data based on the values within our indexed column.

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

A deadlock is essentially a lock that leads to a dead end.

A deadlock occurs when two or more tasks permanently block each other by each task having a lock on a resource which the other tasks are trying to lock. Deadlock is a situation when two processes, each having a lock on one piece of data, attempt to acquire a lock on the other's piece. Each process would wait indefinitely for the other to release the lock, unless one of the user processes is terminated. SQL Server detects deadlocks and terminates one user's process

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

A Livelock is one, where a request for exclusive lock is denied continuously because a series of overlapping shared locks keeps on interfering with each other and to adapt from each other they keep on changing the status which further prevents them to complete the task. SQL Server Livelock occurs when read transactions are applied on a table which prevents write transactions to wait indefinitely. This is different from deadlock as in deadlock both the processes wait on each other.

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

METHOD	DESCRIPTION	EXAMPLE
capitalize()	It capitalises the first letter of the string	print("our example string object".capitalize()) ---> 'Our example string object'
casefold()	It makes all letters in the string lowercase.	print("Our example string object".casefold()) --->'our example string object'
center()	It aligns the string in the centre, using a specified character (space is default) as the fill character.	print('Our example string object'.center(30,'x')) → 'xxOur example string objectxxx'
count()	It counts how many times the character occurs in the string. Note it is case-sensitive.	print("Our example string object".count("r")) ---> 2

endswith()	It searches and confirms if the end of the string ends with the specified character(s). The outcome will be either True or False.	print("Our example string object".endswith("ject")) --->True
find()	It searches the string for a specified value and returns the position of where it was found.	print('Our example string object'.find('m')) ---> 7
format()	It formats specified values in a string	price = 49 print('The bag costs ${}'.format(price)) ---> 'The bag costs $49'
index()	It searches the string for a specified value and returns the position of where it was found	print('Our example string object'.index('string')) ---> 12
isalnum()	It returns True if all characters in the string are alphanumeric.	print('string12'.isalnum()) ---> True

isalpha()	It returns True if all characters in the string are in the alphabet.	print('string’.isalpha()) ---> True
isdigit()	It returns True if all characters in the string are digits.	print('10'.isdigit()) ---> True
islower()	It returns True if all characters in the string are lower case.	print('our example string object'.islower()) ---> True
isnumeric()	It returns True if all characters in the string are numeric.	print('10'.isnumeric()) ---> True
isspace()	It returns True if all characters in the string are whitespaces	print('Our example string object'.isspace()) ---> False
istitle()	It returns True if the string follows the rules of a title	print('Our Example String Object'.istitle()) ---> True
isupper()	It returns True if all characters in the string are upper case.	print('OUR EXAMPLE STRING OBJECT'.isupper()) --->True
join()	It joins the elements of an iterable to the end of the string.	my_list = ['kiwi', 'banana', 'apple'] print(', '.join(my_list)) ---> ‘kiwi, banana, apple’
lower()	It converts a string into lower case.	print('Our example string object'.lower()) --->'our example string object'
lstrip()	It returns a left trim version of the string	print (' Our example string object '.lstrip()) --->'Our example string object '
replace()	It replaces the specified character(s) in the string for another character(s).	print("Our example string object".replace("ject",'jo')) --->’Our example string objo’
rsplit()	It splits the string at the specified separator, and returns a list.	print(‘Our example string object'.rsplit()) ---> ['Our', 'example', 'string', 'object']
rstrip()	It returns a right trim version of the string.	print(' Our example string object '.rstrip()) ---> ' Our example string object'
split()	It splits the string at the specified separator, and returns a list.	print('Our example string object'.split()) ---> ['Our', 'example', 'string', 'object']
splitlines()	It splits the string at line breaks and returns a list.	print('Our example string object'.splitlines()) ---> ['Our example string object']
startswith()	It searches and confirms if the beginning of the string ends with the specified character(s). The outcome will be either True or False.	print("Our example string object".endswith("Ou")) --->True
strip()	It returns a trimmed version of the string.	print(' Our example string object '.strip()) ---> 'Our example string object'
swapcase()	It swaps cases, lower case becomes upper case and vice versa.	print('Our example string object'.swapcase()) ---> 'oUR EXAMPLE STRING OBJECT'

title()	It capitalises the first letter of every word in the string.	print("our example string object".title()) --->'Our Example String Object'
upper()	It capitalises all the letters in the string.	print("Our example string object".upper()) ---> 'OUR EXAMPLE STRING OBJECT'

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

Method	Description	Example
append()	It adds an element at the end of the list.	my_list = ['kiwi', 'banana', 'apple'] my_list.append('pear') print(my_list) --->my_list = ['kiwi', 'banana', 'apple', 'pear']
clear()	It removes all the elements from the list.	my_list = ['kiwi', 'banana', 'apple'] my_list.clear() print(my_list) ---> my_list = []
copy()	It returns a copy of the list.	my_list = ['kiwi', 'banana', 'apple'] print(my_list.copy()) ---> ['kiwi', 'banana', 'apple']
count()	It returns the number of elements with the specified value.	my_list = ['kiwi', 'banana', 'apple', 'kiwi'] my_list.count('kiwi') ---> 2
extend()	It adds the elements of a list (or any iterable), to the end of the current list	my_list = ['kiwi', 'banana', 'apple'] my_favourites = ['pear', 'orange'] my_list.extend(my_favourites) print(my_list) --->my_list = ['kiwi', 'banana', 'apple', 'pear', 'orange']
index()	It returns the index of the first element with the specified value.	my_list = ['kiwi', 'banana', 'apple'] my_list.index('apple') ---> 2
insert()	It adds an element at the specified position.	my_list = ['kiwi', 'banana', 'apple'] my_list.insert(1,'orange') print(my_list) ---> my_list = ['kiwi', 'orange' ,'banana', 'apple']
pop()	It removes the element at the specified position.	my_list = ['kiwi', 'orange' ,'banana', 'apple'] my_list.pop(1) print(my_list) ---> my_list = ['kiwi', 'banana', 'apple']
remove()	It removes the first item with the specified value	my_list = ['kiwi', 'orange' , 'banana', 'apple'] my_list.remove('orange') print(my_list) ---> my_list = ['kiwi', 'banana', 'apple']

reverse()	It reverses the order of the list.	my_list = ['kiwi', 'banana', 'apple'] my_list.reverse() print(my_list) ---> my_list = ['apple', 'banana', 'kiwi']
sort()	It sorts the list, usually in alphabetical order.	my_list = ['kiwi', 'banana', 'apple'] my_list.sort() print(my_list) ---> my_list = ['apple', 'banana', 'kiwi']

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

Method	Description	Example
count()	It returns the number of times a specified value occurs in a tuple.	my_tuple = ('apple', 'banana', 'carrot', 'apple') my_tuple.count('apple') ---> 2
index()	It searches the tuple for a specified value and returns the position of where it was found.	my_tuple = ('apple', 'banana', 'carrot') my_tuple.index('apple') ---> 0

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

Method	Description	Example
clear()	It removes all the elements from the dictionary.	my_dict = {'name':'Shayli','age': 26} my_dict.clear() print(my_dict) ---> my_dict = {}
copy()	It returns a copy of the dictionary.	my_dict = {'name':'Shayli','age': 26} print(my_dict.copy()) ---> {'name': 'Shayli', 'age': 26}
fromkeys()	It returns a dictionary with the specified keys and value.	my_dict = {'name':'Shayli','age': 26} print(new_dict = dict.fromkeys(my_dict)) ---> {'name': None, 'age': None}
get()	It returns the value of the specified key.	my_dict = {'name':'Shayli','age': 26} print(my_dict.get('age')) --->26
items()	It returns a list containing a tuple for each key value pair.	my_dict = {'name':'Shayli','age': 26} print(my_dict.items()) --->dict_items([('name', 'Shayli'), ('age', 26)])
keys()	It returns a list containing the dictionary's keys.	my_dict = {'name':'Shayli','age': 26} print(my_dict.keys()) --->dict_keys(['name', 'age'])
pop()	It removes the element with the specified key.	my_dict = {'name':'Shayli','age': 26} print(my_dict.pop('name')) print(my_dict) --->'Shayli' {'age': 26}
popitem()	It Removes the last inserted key-value pair.	my_dict = {'name':'Shayli','age': 26} print(my_dict.popitem()) print(my_dict) --->('age', 26)

		{'name':'Shayli'}

setdefault()	It returns the value of the specified key. If the key does not exist: insert the key, with the specified value.	my_dict = {'name':'Shayli','age': 26} print(my_dict.setdefault('name')) --->Shayli
update()	It updates the dictionary with the specified key-value pairs.	my_dict = {'name':'Shayli','age': 26} my_dict.update({'age': 27}) print(my_dict) --->{'name': 'Shayli', 'age': 27}
values()	It returns a list of all the values in the dictionary.	my_dict = {'name':'Shayli','age': 26} print(my_dict.values()) ---> dict_values(['Shayli', 26])

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

Method	Description	Example
add()	It adds an element to the set.	my_set = {'apple', 'banana' , 'cherry'} my_set.add() print(my_set) --->{'banana', 'cherry', 'pear', 'apple'}
clear()	It removes all the elements from the set.	my_set = {'apple', 'banana' , 'cherry'} my_set.clear() print(my_set) ---> {}
copy()	It returns a copy of the set.	my_set = {'apple', 'banana' , 'cherry'} print(my_set.copy()) ---> {'banana', 'cherry', 'apple'}
difference()	It returns a set containing the difference between two or more sets.	my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'} print(my_set.difference(set2)) ---> {'banana', 'cherry', 'apple'}
intersection()	It returns a set, that is the intersection of two or more sets.	my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'} print(my_set.intersection(set2)) --->{'pineapple', 'mango'}
issubset()	It returns whether another set contains this set or not.	my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'} print(set2.issubset(my_set)) --->True
issuperset()	It returns whether this set contains another set or not.	my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'} print(my_set.issuperset(set2)) --->True

pop()	It removes an element from the set.	my_set = {'apple', 'banana' , 'cherry'} my_set.pop() print(my_set) --->my_set = {'cherry', 'apple'}
remove()	It removes the specified element.	my_set = {'apple', 'banana' , 'cherry', 'pear'} my_set.remove('pear') print(my_set) --->{'banana', 'cherry', 'apple'}
symmetric_difference()	It returns a set with the symmetric differences of two sets.	my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'} print(my_set.symmetric_difference(set2)) --->{'cherry', 'apple'}
union()	It returns a set containing the union of sets.	my_set = {'apple', 'banana' , 'cherry'} set2 = {'mango', 'pineapple'} print(my_set.union(set2)) print(my_set) print(set2) --->{'banana', 'pineapple', 'mango', 'cherry', 'apple'} {'apple', 'banana' , 'cherry'} {'mango', 'pineapple'}

update()	It updates the set with another set, or any other iterable.	my_set = {'apple', 'banana' , 'cherry'} set2 = {'mango', 'pineapple'} my_set.update(set2) print(my_set) print(set2) --->my_set = {'banana', 'pineapple', 'mango', 'cherry', 'apple'} set2 = {'mango', 'pineapple'}

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

Method	Description	Example
read()	It returns the file content.	f = open("demofile.txt", "r") print(f.read())
readline()	It returns one line from the file.	f = open("demofile.txt", "r") print(f.readline( ))
readlines()	It returns a list of lines from the file.	f = open("demofile.txt", "r") print(f.readlines())

write()	It writes the specified string to the file.	f = open("demofile.txt", "a") f.write("Hello CFG!") f.close()
writelines()	It writes a list of strings to the file.	f = open("demofile3.txt", "a") f.writelines(["Hello CFG!", "This is my Theory Assessment"]) f.close()