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 programming language, that is high level and highly dynamic. Its main features include a human-readable syntax, built in data structures, and numerous libraries and modules encouraging code reusability.

2. Discuss the difference between Python 2 and Python 3

Python 3 is the newer version of the python language compared to python 2. Here are some of the differences between them:

- **Integer division**: The calculation 5/2 would result in 2, another integer, in python2, however it would return 2.5, a float, in python3.
- **Stability**: Python2 is considered more stable compared to python3 due to its legacy and long-standing history.
- **Print**: In python2, the print is a statement, e.g. print "hello", versus the print function in python3, e.g. print("hello").
- **String types:** python2 has two string types, Unicode string and non-Unicode string, whereas in python3, all strings are Unicode strings by default.

3. What is PEP 8?

PEP8, Python Enhancement Proposal, is a document created that includes the best

practices and guidelines in writing clean python code, with a focus on readability and consistency of python code. An example could include the way we name our variables or functions, i.e. by separating words with an underscore, e.g. myFunction should be written as my_function, whereas a class should be written in camel case, where every word starts in a capital letter – and no underscore between the words, e.g. MyClass.

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

A program is essentially a sequence of instructions for a computer, which can be built to execute algorithms.

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

A process is an instance of a computer program being executed by one or more threads, a program running could have multiple processes, and these processes could spawn children. The key feature of processes is that they die.

6. In computing / computer science what is cache?

The cache is a part of the CPU that is used to temporarily hold a segment of instructions to be reused in order to make instruction retrieval from the RAM more efficient for the CPU.

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

A thread is a set of instructions that are used by a program that can be run simultaneously, i.e. multi-threading in order to run parallel tasks. An example of this is having a thread that takes care of the graphical user interface that a user sees on the screen, while another thread takes care of the calculations behind the scenes to be returned to the user.

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

Concurrency is a CPU's ability to work on multiple processes by switching between them at such a high frequency that it is pretty much concurrent, whereas **parallelism** is a way to run multiple tasks on the same data.

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

GIL, or Global Interpreter Lock, in python refers to a lock that controls and limits to only one thread to be in control of the python interpreter, essentially it works by preventing two threads from executing within the same program as one will hold the 'lock'.

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

- **DRY:** Don't Repeat Yourself – i.e. do not re-write code that can be referenced or reused in other areas, therefore keeping your code efficient, readable and clean all in order to reduce redundancies.
- **KISS:** Keep It Simple, Stupid – i.e. this principle states that the best solutions

are the simplest, in that solutions should aim to be as simple as possible and not over complicated, especially seen in designing systems with abstractions and inheritance.
- **BDUF:** Big Design Up Front – i.e. a software development approach in which the design of a program or system is to be completed before the system itself is created, seen in more rigid type of software development models, such as the waterfall model.

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

The garbage collector in python is a way that the language automatically handles and disposes of objects in memory when we are writing our code in order to free up memory. It works by running during the programs execution, finding objects that lose its reference pointer, or its reference goes out of scope, and then it will take that orphaned object and destroy it, thereby cleaning up memory.

12. How is memory managed in Python?

Python uses a garbage collector to manage memory so that users do not have to manually collect it. It essentially collects and frees up memory to be used and made available for other objects. Reference counting is the technique used in which objects are deallocated memory space when there is no reference or pointer to it anymore, e.g. a variable is lost and unreferenced in a code. These are two ways that python manages memory.

13. What is a Python module?

A module in python refers to a file that contains statements and definitions, which could define functions, classes and variables. They are used to bread down large issues into organized files in order to provide solutions.

14. What is docstring in Python?

Docstrings are the string text we leave in python code that represent the documentation of the python code we are working on. They are specifically string comments that are written right after the definition of a method, class or module, and it can accessed using the __doc__ attribute in order to get readable information on any method, class or module.

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

Pickling is the process where a python object hierarchy is converted into a byte stream, essentially serialization, and unpickling is the opposite of that. An example of pickling and unpickling usages include when sending objects over a TCP connection, or storing python objects in a database, or caching objects in a python dictionary.

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

There are some tools that can help to find bugs or perform static analysis such as pychecker and pylink. They can be used to warn about the style and complexity of the bug, mostly compile-time bugs. The one downful is that they can't find runtime bugs as they will occur during the runtime of the program.

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

All arguments passed in python are passed by reference, meaning that the pointer or reference to that variable is what is passed in, one advantage of this is that is is fast as there is no need for another copy to be made as only the reference is passed through. An example of this includes an example of this is creating a function that adds to a list and because the reference of the argument is passed into the calling of the function, then all the lists will get the same value appended.

```
Def func(x):
        x.append("added")
list_1 = []
func(list_1)
print(list_1)  # prints ["added"]
list_2 = []
func(list_2)
print(list_2)  # also prints ["added"]
```

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

Using dictionary or list comprehension is a much faster and easier way of creating or filling in a dictionary or list with values, but also you can use it to perform actions or filtering onto the values in a list or dictionary to be saved to a new variable. This is all done during the creation of a new dictionary or list. E.g.

List_1 = [i * 2 for i in range(5)] # will create this list [0, 2, 4, 6, 8]

Dict_1 = {i:i*2 for i in List_1} # will create this dictionary {0: 0, 2: 4, 4: 8, 6: 12, 8: 16}

19. What is namespace in Python?

A namespace essentially is a collection of names. They can be local, i.e found in functions, or global, i.e. they sit in a module (file), or in the built-in namespace, ushc as the print() function name. So when a reference is made inside a function, it is searched in the local namespace, then in the global, then in the built-in and if nothing with that name exists within the same namespace, then it can be created- but it also means that any changes to a variable with that same name in the local will be changed if it is also changed in the global, such as this example found online:

```
def outer_function():
    a = 20

    def inner_function():
        a = 30
        print('a =', a)

    inner_function()
    print('a =', a)


a = 10
outer_function()
print('a =', a)
```

20. What is pass in Python?

A pass is a null statement, which isn't ignored by the interpreter, it just means that there is no result to that statement when run. It is useful when creating new functions or classes and write pass into it to be able to use the file but not worry about any errors, so that you can add implementation later.

21. What is unit test in Python?

A unit test is used to test individual units of source code in order to determine if the code functions as it is required to do so. We try to test many small units of code such as. A few tests for every function in order to test for edge cases to preemptively correct for any human errors in our program.

22. In Python what is slicing?

Slicing in python refers to a way of accessing parts of a sequence such as strings, tuples and lists, and you can also use them to modify mutable sequences such as lists. You can specify the start and end of the slice you want to make, and you can also specify the step i.e. slice only every other item.

23. What is a negative index in Python?

Negative index refers to a way to access the index from the opposite end, i.e. from the last elements. If I wanted to access the last element in a list I could use negative index instead of needing to know exactly how many elements there are to then access the last item on a list, e.g.:

```
List_1 = [1, 2, 3, 4, 5]
Last_item1 = List_1[4] # I can access it directly because I know that there are 5 items in it
Last_item2 = List[-1] # Returns the last item i.e. 5
```

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

A ternary operator is a way to write conditional statements in a shortened way that will be evaluated also as true or false. It is written with an **if** and an **else**, e.g.:

```
Age = 17
Legal_age = True is Age >= 21 else False
Print(Legal_age) # will print False
```

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

*args and **kwargs are used to allow an unknown number of arguments or keyword arguments to be passed through a function when called. The difference between *args and ** kwargs is that *args will allow a number of raw values, named arguments, to be inserted into a function call to be passed into it, such as:

```
Def sum(*args):
        result = 0
        for x in args:
                result += x
        return result

sum(1, 2, 3) # will return 6
sum(6, 7, 8, 9, 10, 11, 12) # will return 63
```

On the other hand, **kwargs represents keyword arguments or named arguments that can be used within the function. It is unpacked as a dictionary. Example:

```
def sum_salaries(**kwargs):
        result = 0
        for arg in kwarg.values():
                result += arg
        return result

print(sum_salaries(Amy=23000, Mark=12000, Ruth=20000)) # will print 55000
```

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

Range returns a range object that is iterable, whereas the xrange returns a generator object that can be used to display numbers by looping. Essentially creating two variables using range and xrange, the range variable will be a list object that contains a range of numbers whereas the xrange variable will return an xrange object. Xrange saves more memory compares to range. However, in python3, there is no xrange, only range, essentially python2's xrange has now become python3's range. So now using the range function in creating a variable will now return a range object.

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

Flask is a web framework that provides us with tools and technologies that allow us to build web applications. It is a very popular web framework and is considered more pythonic that other frameworks such as Django. It is also beginner friendly and easy to setup. It can be used to create Flask applications such as websites, but also with other technologies such as Docker and React in creating microservices, or use it with javascript and twilio to create a video chat application.

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

A clustured index is an index in where the table is reordered physically in the table, whereas a non-clustuered index is an index in which a separate table is created, then ordered and has pointers to the location of the rows to the rows in the original table. The key difference between the two is that you can only have one clustered index, whereas you can have multiple non-clustured indexes in a single table.

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

A deadlock refers to the situation where two or more transactions are waiting indefinitely for one another to give up the lock in order to make changes to data in a table. The database management system should detect deadlocks and will deal with it in one of two ways, the **wait-die scheme**, or the **wound wait scheme**, where the first scheme will check the timestamps of the transactions involved and allow one of the transactions to wait until the resources that it needs become available where it may need to be 'killed' and then revived with a minute delay with the same timestamp in order to be able to recerive the resources it requires, essentially the scheme will allow the older transaction to wait but will kill the younger one. On the other hand, the second scheme, gives the transactions the power to kill the younger transaction or wait for the older transaction until they release the resources they need.

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

Livelock refers to a situation where a request, process or transaction is denied the request for the lock through continuous overlapping shared locks that keep interfering with each other.

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

METHOD	DESCRIPTION	EXAMPLE
capitalize()	This method will capitalize the first letter of the first word in a string.	txt = "silence of the lambs" capitalised_txt = txt.capitalize() print(capitalised_txt) # prints 'Silence of the lambs'
casefold()	This method will change all the characters in a string into lower case.	txt = "Hello, And Welcome To My World!" x = txt.casefold() print(x) # prints 'hello, and welcome to my world'
center()	This method will position any string in the center of however many character spaces specified in the argument passed into it when called.	txt = "apple" spaced = txt.center(10) print(spaced) # prints ' apple '
count()	This will count the number of times a certain string, the argument passed through, is found within another string.	txt = "I love apples, and I love oranges!" count = txt.count("love") print(count) # prints 2
endswith()	This will return true is a string ends with a specific value that is passed as an argument to it.	txt = "I love apples, and I love oranges!" endsWith = txt.endswith("!") print(endsWith) # prints True
find()	The find method will return the index of the first occurrence of the string passed into it as an argument but will return -1 if the string isn't found within the wider string.	txt = "I love apples, and I love oranges!" found = txt.find("pears") print(found) # prints -1
format()	The format method can be used as a placeholder to insert values into a string as a pre-specified format.	txt "They cost {price:.2f} pounds!" print(txt.format(price = 20)) # prints They cost 20.00 pounds
index()	This method will return the index of the first occurrence of a specified value passed as an argument in the method call similar to the fine() method, however the difference here is that it will raise an exception if the value isn't found in the wider string, whereas the find() method will return -1.	txt = "I love apples, and I love oranges!" index = txt.index("pears") print(index) # raises an exception, specifically a ValueError: substring not found
isalnum()	This method will return True if the string its called on is alphanumeric, however if it contains any value that is not alphanumeric such as -+'\][;., then it will return False.	txt = "Hello world" isAlphanumeric = txt.isalnum() print(isAlphanumeric) # prints True

isalpha()	This method will only return True if all the characters in the string it's called on are found within the alphabet, i.e. a-z A-Z.	txt = "Hello!" isAlpha = txt.isalpha() print(isAlpha) # prints False
isdigit()	This will return true only if the string it is called on is a string of digits otherwise it will be false.	txt = "500" isDigit = txt.isdigit() print(isDigit) # prints True
islower()	This will only return True if all the characters in a string it is called on are lower case, otherwise it will return false.	txt = "hello world" isLower = txt.islower() print(isLower) # prints True
isnumeric()	This will only return True if all the characters in a string it is called on are numeric, otherwise it will return false.	txt = "555" x = txt.isnumeric() print(x) # return True
isspace()	This will only return True if all the characters in a string it is called on are whitespaces, otherwise it will return false.	txt = " - " x = txt.isspace() print(x) # return False
istitle()	This method will return True only is the words within a string are all capitalized, otherwise it will return false.	txt = "Hello how are you?" print(txt.istitle()) # will print False
isupper()	This method will True if all the characters in a string are in uppercase, and False otherwise.	txt = "HELLO!" print(txt.isupper()) # will print True
join()	This method will join the values held in an iterable object such as a list or tuple, and it will fill in between the values with whatever it is passed on, and the iterable object is passed through it as an argument.	names = ["Amy", "Maria", "Ruth"] print("*".join(names)) # will print 'Amy*Maria*Ruth'
lower()	This will return a string that is all lower case. It does the same as casefold() in English, however using lower() and casefold() with characters of other language that use more than the english 26 letter ASCII charcters, show that casefold is more aggressive and will convert more characters into lowercase english characters.	txt = "HELLO world" print(txt.lower()) # prints 'hellow world'
lstrip()	This method will remove any characters passed into it from the string it is called on, however if no characters are passed into it as arguments, then it will remove any whitespces leading up to the text in a string.	txt = '.....hello world' print(txt.lstrip('.')) # will print 'hello world'
replace()	This method takes in two arguments, the first is the word to be replaced and the second is the word replacing it.	txt = "I like apples" changed_txt = txt.replace("apples", "pears") print(changed_txt) # this will now print 'I like pears'
rsplit()	This method will split a string on specific characters passed as its arguments, if nothing is passed into it, it will split the string on the whitespaces. The split words are returned as a list of words.	txt = "apples, bananas, pears" split_fruit = txt.rsplit(", ") print(split_fruit) # will print ["apple", "bananas", 'pears"]
rstrip()	This method will remove the trailing characters at the end of a string, if no character is specified whitespaces are removed.	txt = "I like apples....." strip_txt = txt.rstrip('.') print(strip_txt) # will print 'I like apples'
split()	This method will split a string on a specified character or on whitespaces if nothing is specified.	txt = "I like apples" split = txt.split() print(split) # will print ["I", "like", "apples"]
splitlines()	The splitlines method will split a string on the new line code '\n' and returns a list of the separate lines as separate values.	txt = "I like apples\nI like pears" split_lines = txt.splitlines() print(splint_lines) # this will print a list ["I like apples", "I like pears"]
startswith()	This will return true if a string starts with a specific sequence of characters in a wider string.	txt = "I like apples" print(txt.startswith("I like") # will print True

strip()	The strip method will remove both leading and trailing spaces, or pre-specified characters.	txt = " apples. " print(txt.strip()) # will print "apples."
swapcase()	This method will swap the cases of the characters in a string from lowercase to uppercase and the opposite too.	txt = "I live in Birmingham" print(txt.swapcase()) # will print "i LIVE IN bIRMINGHAM"

title()	The title method will capitalize every word in a string.	txt = "I like apples and oranges" print(txt.title()) # will print "I Like Apples And Oranges"
upper()	This method will convert every letter character in a string into uppercase	txt = "I like apples and oranges" print(txt.upper()) # will print "I LIKE APPLES AND ORANGES"

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

Method	Description	Example
append()	This method will add an element to the end of a list	colours = [‟red″, ‟blue″] colours.append(‟yellow″) print(colours) # will print [‟red″, ‟blue″, ‟yellow″]
clear()	The clear method will remove all the element held within a list	colours = [‟red″, ‟blue″] colours.clear() print(colours) # will print []
copy()	The copy method will copy the elements of one list into a new list to be saved into a variable.	colours = [‟red″, ‟blue″] copy = colours.copy() print(copy) # will print [‟red″, ‟blue″]
count()	This method will the number of times an element is found in a list.	colours = [‟red″, ‟red″, ‟blue″] x= colours.count(‟red″) print(x) # will print 2
extend()	This will add elements of any iterable object to the end of a current list.	colours = [‟red″, ‟blue″] more_colours = [‟yellow″, ‟pink″] colours.extend(more_colours) print(colours) # will print [‟red″, ‟blue″, ‟yellow″, ‟pink″]
index()	This method will return the index of the first occurrence of a specified element, will throw an error if it is not found in the list.	colours = [‟red″, ‟blue″] print(colours.index(‟red″)) # will print 0
insert()	The insert method takes in two arguments, the first is the position of the index to insert into, and the second if the element that will be added to the list at that index.	colours = [‟red″, ‟blue″] colours.insert(1, ‟yellow″ print(colours) # will print [‟red″, ‟yellow″, ‟blue″]
pop()	The pop method is used to remove an element from a specific position that is passed as an argument, if no index number is passed through, then the last element is removed.	colours = [‟red″, ‟blue″] colours.pop() print(colours) # will print [‟red″]
remove()	This method will remove the first occurrence of a specified value that is passed as an argument. It must take an argument or else it will throw an error.	colours = [‟red″, ‟blue″] colours.remove(‟red″) print(colours) # will print [‟blue″]
reverse()	This method will simply reverse the order of elements held within a list.	colours = [‟red″, ‟blue″] colours.reverse() print(colours) # will print [‟blue″, ‟red″]

sort()	This method will sort the elements into an order based on the data type it holds.	colours = [‟yellow″, ‟red″, ‟blue″] colours.cort() print(colours) # will print [‟blue″, ‟red″, ‟yellow″]

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

Method	Description	Example
count()	The count method will return the number of occurrences of a specific value within a tuple.	X = (1, 2, 6, 4, 8, 4) Y = X.count(4) print(Y) # will print 2
index()	This method will return the index of the first instance of a specific value that is passed through as an argument, if however the value isn't found in the tuple, then an error is raised.	X = (1, 2, 6, 4, 8, 4) Y = X.index(4) print(Y) # will print 3

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 a dictionary.	fruits = {"apple": "red", "banana": "yellow"} fruits.clear() print(fruits) #prints {}
copy()	This method will copy a dictionary to be saved into another variable.	fruits = {"apple": "red", "banana": "yellow"} more_fruits = fruits.copy() print(more_fruits) #prints {"apple": "red", "banana": "yellow"}
fromkeys()	This method will create a new dictionary based on specific values that will form the keys and specific values that will form the values of those keys.	keys = (1, 2, 3) values = "a" dictionary = dict.fromkeys(keys, values) print(dictionary) # will print {1: "a", 2: "a", 3: "a"}
get()	This will get and return the value of a specific key within a dictionary, if you enter an invalid key then it will return None.	fruits = {"apple": "red", "banana": "yellow"} apple_colour = fruits.get("apple") print(apple_colour) # will print "red"
items()	This will return a dictionary view object that contains the key and values in tuple pairs held within a list. It will also reflect any and all changes made to the dictionary.	fruits = {"apple": "red", "banana": "yellow"} x = fruits.items() print(x) # will print dict_items([('apple', 'red'), ('banana', 'yellow')])
keys()	This will return a list of all the keys that are in a dictionary.	fruits = {"apple": "red", "banana": "yellow"} keys = fruits.keys()

		print(keys) # will print [“apple”, “banana”]
pop()	This method will remove a specific element in a dictionary based on the key that is passed into it. If no key is passed through, it will raise an error.	fruits = {“apple”: “red”, “banana”: “yellow”} fruits.pop(“apple”) print(fruits) # will print {“banana”: “yellow”}
popitem()	This method will remove the last key and value pair in a dictionary.	fruits = {“apple”: “red”, “banana”: “yellow”} fruits.popitem() print(fruits) # will print {“apple”: “red”}

setdefault()	The setdefault() method will return the value of an item with that specific key, however if that key does not exist already in the dictionary, then it will be added with the value passed alongside it.	fruits = {"apple": "red", "banana": "yellow"} default = fruits.setdefault("pear", "green") print(fruits) # will print {"apple": "red", "banana": "yellow", "pear": "green"}
update()	This will update the dictionary by passing through a key and value pair wrapped in {}. If the key already exists, then the value will be updated, otherwise the key and value pair will be added to the end of the dictionary.	fruits = {"apple": "red", "banana": "yellow"} fruits.update({"apple": "pink"}) print(fruits) # will print {"apple": "pink", "banana": "yellow"}
values()	This method will return a list of the values of a dictionary.	fruits = {"apple": "red", "banana": "yellow"} values = fruits.values() print(values) # will print ["red", "yellow"]

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

Method	Description	Example
add()	This method will add a value to a set. If the value added is already in the set then it will be considered as already added as a set cannot contain multiple of the same value.	fruits = {"apple", "banana"} fruits.add("pear") print(fruits) # will print {"apple", "banana", "pear"}
clear()	This will clear and remove all the elements within a set.	fruits = {"apple", "banana"} fruits.clear() print(fruits) # will print set()
copy()	This method will return a copy of a set to be saved into another variable.	fruits = {"apple", "banana"} all_fruits = fruits.copy() print(fruits) # will print {"apple", "banana"}
difference()	This method will return all the elements that are unique to the set it is called on compared to the set passed through the method parameters.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"} differences = fruits.difference(smoothies) print(differences) # will print {"apple", "pears"}
intersection()	This method will return a set with the shared elements of the set the method is called on and the set the method has passed in its parameter.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"}

		intersect = fruits.intersection(smoothies) print(intersect) # will print {"banana"}
issubset()	This method will check if the elements of the set its called on are all found within the set that is passed through the method parameters. If so, then it returns True, otherwise it will return False.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"} is_subset = fruits.issubset(smoothies) print(is_subset) # will print False
issuperset()	This will do the inverse of the issubset method, where it checks if the set passed into the parameters elements' are contained in the set the method is called on.	fruits = {"apple", "banana", "pears"} smoothie_fruits = {"banana", "apple"} is_super_set = fruits.issuperset(smoothie_fruits) print(is_super_set) # will print True
pop()	This will remove the last element of a set, it does not take an argument, and will throw an error if used on an empty set.	fruits = {"apple", "banana", "pears"} fruits.pop() print(fruits) # will print {"apple", "banana"}
remove()	This method will remove an element from a set by passing that element through as an argument. It will however raise an error if that element does not exist in the set.	fruits = {"apple", "banana", "pears"} fruits.remove("apple" print(fruits) # will print {"banana", "pears"}
symmetric_difference()	This will return a set full of all the unique differences between two sets, i.e. the elements they do not share together.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"} non_shared_elements = fruits.symmetric_difference(smoothies) print(non_shared_elements) # will print {"apple", "pears", "peanut butter", "chia seeds"}
union()	This will return a union of the two sets passed into and on this method, where one copy of every element is copied into a new set.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"} fruit_smoothies = fruits.union(smoothies) print(fruit_smoothies) # will print {"apple", "banana", "pears", "peanut butter", "chia seeds"}

update()	This method will add elements that a set doesn't already contain from another set that has been passed through as an argument.	fruits = {"apple", "banana", "pears"} smoothies = {"banana", "peanut butter", "chia seeds"} fruits.update(smoothies) print(fruits) # will print {"apple", "banana", "pears", "peanut butter", "chia seeds"}

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

Method	Description	Example
read()	This method will return all the contents found within a file, you can specify how many bytes to 'read' from a file to be returned and saved into a variable.	file = open("example.txt", "r") print(file.read()) # will print the text found in this file
readline()	This will read one line from a file. One line will end at "\n".	file = open("example.txt", "r") print(file.read()) # will print the first line found in this file
readlines()	This method will read and return all the lines within a file inside a list.	file = open("example.txt", "r") print(file.read()) # will print a list containing each line of a file.
write()	This method will 'write' to a file by adding the string passed into the write method to the file it is called on.	file = open("example.txt", "a") file.write("Adding this line!") file.close() file = open("example.txt, "r") print(file.read()) # will print the text found in this file with the newly added line

writelines()	This method will write a list of strings to the end of a file.	file = open("example.txt", "a") file.write([‟Adding this line!‟, ‟Adding this line too‟, ‟Final sentence!‟]) file.close() file = open(‟example.txt, ‟r‟) print(file.read()) # will print the text found in this file with the newly added lines.