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 that is considered easy to learn thanks to its easy to read look. Python's main features include open source libraries, easy automation, community support, object-oriented, scalable, portable and is a cross platform language.

2. Discuss the difference between Python 2 and Python 3

Python 3 is the newer version of the language. Python 2 is the legacy version. The main differences include some libraries that are not compatible with python 3. In python 2 strings are stored in ASCII, python 3 is unicode by default. Calculations are slightly different, with python 3 returning the expected result. Some syntax is also slightly different.

3. What is PEP 8?

PEP 8 is a document that helps us to provide guidelines on how to write python. It provides best practices with the aim to enhance readability and consistency in the code. It can be found here: https://peps.python.org/pep-0008/

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

A program provides the computer with coded instructions on how to perform a task.

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

A process refers to the set of instructions that are currently being processed by the computer processor. It is the program running in the computer.

6. In computing / computer science what is cache?

It can be a hardware or software component that stores data so that future requests for that data can be served faster. It is a high-speed data storage layer.

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

A thread is a small set of instructions that run on a schedule and are executed independently by the CPU. It runs within a program. A single thread has a beginning, sequence and end and at any given time there is a point of execution.
Multithreading refers to multiple threads that run at the same time and perform different tasks in a single program.

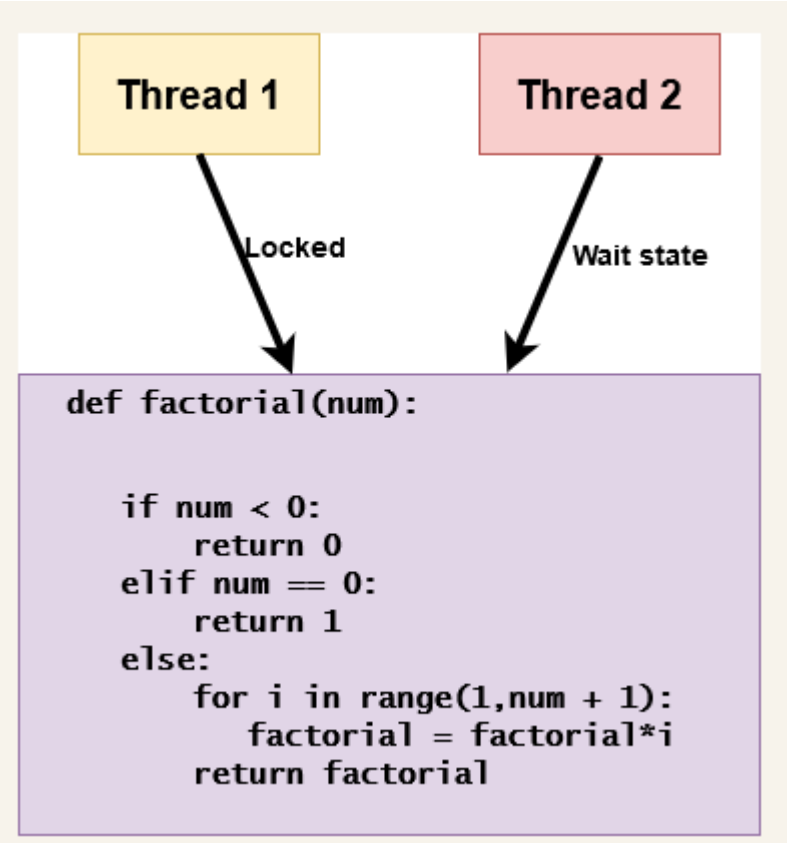
8. In computing / computer science what is concurrency and parallelism and what are the differences?

Concurrency - is when multiple tasks can run in overlapping periods. It gives the illusion of multiple tasks running at the same time due to it's fast switching by the CPU.
Concurrency is achieved by using threading, while parallelism is achieved by using multitasking.
Parallelism - is when tasks actually run in parallel in multiple CPUs.

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

Global Interpreter Lock - is a lock that allows only one thread to hold control of the interpreter. Therefore, only one thread can be executed at any point in time.
The diagram shows that only one of the threads is able to access the function at

any one time. Thread 2 is in wait state while thread 1 is locked.



Source: https://www.datacamp.com/tutorial/python-global-interpreter-lock

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

DRY- Don't Repeat Yourself principle aims to reduce the repetition of patterns and code duplication by favoring data normalization or abstractions to avoid redundancy.

KISS- Keep It Simple Stupid is a design principle that encourages keeping designs/ systems as simple as possible. It focuses on the idea that if we can't understand a product, we can't use it properly and that the widest possible audience must be able to understand it, if the product is to gain maximum market share - it was first used for product design but the same principles can be transferred to software development.

BDUF- Big Design Upfront is an approach in which the program's design should be completed, developed and perfected before implementation is started. Initial detailed design is created before coding and testing takes place. Its principles include - think before you code, support governance and abstract thinking.

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

It is a form of memory management that is automated within Python. Garbage collection is the process by which python periodically frees and reclaims blocks of memory that are no longer in use to free up memory space. It has an algorithm to remove objects that are no longer needed using reference counting or generational Garbage Collection. 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 where reference count doesn't reach zero, you wait for the generational garbage collection algorithm to run and clean the object.

12. How is memory managed in Python?

In Python memory allocation and removal methods are automatic and managed by the interpreter. In python whenever a variable is assigned a value, the python memory manager will check if an object with that value is already available in the memory . If an object is already present in memory , then this variable points to that object instead of creating a new object with the same value. If an object with that value is not available in memory (heap) , python memory manager will create a new object with the specified value and this variable will point to this newly created object on heap(memory)

Stack Memory —all the methods/method calls , references are stored in stack memory.
Heap Memory — all the objects are stored in heap

Python uses below 2 algorithms for garbage collection:
- **Reference counting**
- **Generational garbage collection**

13. What is a Python module?

A module is like being a code library - A file containing a set of functions you want to include. To create a module you must save the code you want in a file with the file extension .py, You can name the module file whatever you like. To use the module you can use the import statement:

```
#create a module
def greeting(name):
    print("Hello, " + name)
#use a module
 import mymodule
mymodule.greeting("Claire")
```

14. What is docstring in Python?

A docstring as string literals that appear after a definition of a function, method, class or module. They are used to document our code, access them using the __doc__ attribute.

```
def square(n):
    '''Takes in a number n, returns the square of n'''
    return n**2
```

Inside the triple quotation marks is the docstring of the function square( ) as it appears right after its definition.

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

Pickling - a python object hierarchy is converted into a byte stream
Unpickling - is the opposite, a byte stream is converted back into an object hierarchy.

```
#pickle a list
import pickle
mylist = ['a', 'b', 'c', 'd']
with open('datafile.txt', 'wb') as fh:
  pickle.dump(mylist, fh)
```

Output: A new file named "datafile.txt" is created, which converts the mylist data in the byte stream.

```
#unpickle a list
import pickle
pickle_off = open ("datafile.txt", "rb")
emp = pickle.load(pickle_off)
print(emp)
```

Output: mylist data ['a', 'b', 'c', 'd']

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

Pychecker - is an open source tool for static analysis that detects bugs from source code, gives warnings about style and complexity of the bug.

Pylint- also open source, that looks for programming errors and can be used for coding standards. It can check the length of each line and variable names to project style.

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

Python's argument model is neither "Pass by Value" or "Pass by Reference" but it is "Pass by Object Reference".

'Pass by reference' means that you have to pass the function(reference) to a variable which means that the variable already exists in memory. When we pass something by reference, then any change we make to the variable inside the function, those changes are reflected to the outside value as well:

```
student = {'Jim': 12, 'Anna': 14, 'Preet': 10}
def test(student):
  new = {'Sam':20, 'Steve':21}
  student.update(new)
  print("Inside the function", student)
  return
test(student)
print("Outside the function:", student)
```

Output:
Inside the function {'Jim':12, 'Anna':14, 'Preet':10, 'Sam':20, 'Steve': 21}
Outside the function:{'Jim':12, 'Anna':14, 'Preet':10, 'Sam':20, 'Steve': 21}

In 'pass by value' we pass a copy of actual variables in function as a parameter. Any modifications made to the parameter will not be reflected in the variable. Therefore, changes made are not reflected back to the calling function:

```
student = {'Jim': 12, 'Anna': 14, 'Preet': 10}
def test(student):
  student = {'Sam':20, 'Steve':21}
  print("Inside the function", student)
  return
test(student)
print("Outside the function:", student)
```

Output:
Inside the function {'Sam':20, 'Steve': 21}
Outside the function: {'Jim':12, 'Anna':14, 'Preet':10}

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

List comprehension - creates a shorter syntax for when you want to create a new list based on the values in an existing list.

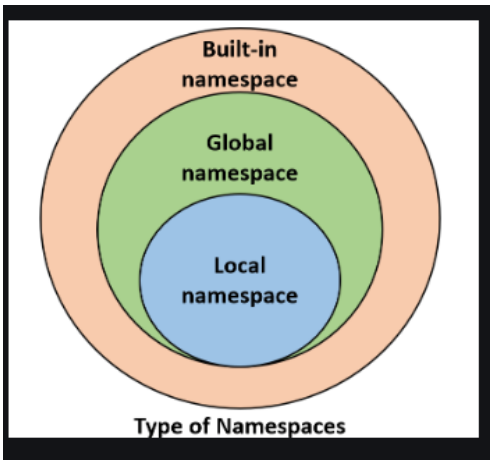
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x] *#list comprehension*
print(newlist)

Dictionary Comprehension - same concept, transforms one dictionary into another in one line syntax. It is a powerful concept and can be used to substitute for loops and lambda functions.

 # dictionary comprehension example
square_dict = {num: num*num for num in range(1, 11)}
print(square_dict)

19. What is namespace in Python?

A namespace is a system that has a unique name for each and every object in Python, this might be a variable or a method. It ensures each object can be used without causing conflict. The diagram below visualizes the types of namespaces and their scope.



20.What is pass in Python?

A placeholder for future code. Nothing will happen when the pass statement is executed. You can avoid getting an error when empty code is not allowed. Syntax:

```
def myfunction():
    pass
```

21. What is unit test in Python?

Unit testing is a method for testing that looks at the smallest testable pieces of code, called units, which are tested for correct operation. By doing unit testing, it can verify that each part of the code works correctly and as predicted.

22. In Python what is slicing?

Slicing involves cutting part of the string to return a part of it, using the indexes. See example below: It returns the string that is sliced from index to 2 - 5.

```
b = "Hello, World!"
print(b[2:5])
```

Output: llo

23. What is a negative index in Python?

We ca use negative indexes to start the slice from the end of the string, by using a negative number it starts the slicing in reverse:

```
b = "Hello, World!"
print(b[-5:-2])
```

Output: orl

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

Ternary operators can also be called conditional expressions, these are operators that check something based on a condition being true or false. It simply allows testing a condition in a single line replacing the multiline if-else making the code compact and quicker.

```
age = 33
discount = True if age >= 65 else False
print(discount)
```

Output:False

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

***args - stands for Arbitrary Arguments. If you do not know how many arguments will be passed into your function we can add the ' * ' This way the function will receive a tuple of arguments, and can access the items accordingly.**

```
def my_function(*pets):
  print("The youngest pet " + pets[2])

my_function("Cat", "Dog", "Horse")
```

Output: Horse

****kwargs - similar to args except it is used when you do not know how many keyword arguments that will be passed into your function, add two asterisk:' ** '**
This way the function will receive a dictionary of arguments, and can access the items accordingly

```
def my_function(**pet):
  print("His first name is " + pet["fname"])

my_function(fname = "Attie", lname = "Manu")
```

Output: His first name is Attie

26. How are range and xrange different from one another?
Used to generate a list of integers. Two functions that can be used to iterate a certain number of times in for loops.

range - This returns a range object (a type of iterable) All arithmetic operations can be performed as it returns a list - this is not possible with xrange.

xrange - This function returns the generator object that can be used to display numbers only by looping. Xrange is faster and uses less memory than range.

```
n = xrange(10)
print(list(n))
Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

The return type of range() is :
<type 'list'>
The return type of xrange() is :
<type 'xrange'>
```

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

Flask is a web framework that provides tools to assist making web apps easier in python. It is useful for developers and new users since it allows you to build a web app quickly using only a single python file. It comes with some standard functionalities and allows developers to add any number of libraries or plugins for an extension, making it a popular developer tool.

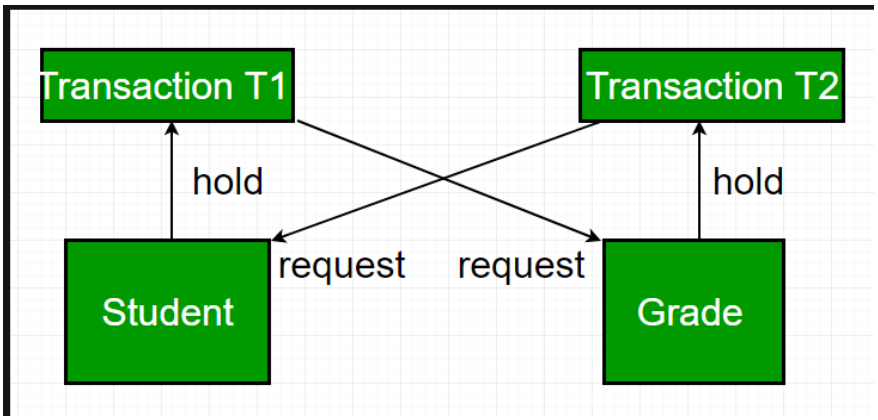
28. What are clustered and non-clustered index in a relational database?

These are two types of indexes in SQL server.
Clustered index - defines the order in which data is physically stored in a table. There can be only one clustered index per table. The primary key constraint auto creates a clustered index on that particular column. It only works if the data or file that you are moving into secondary memory should be in sequential or sorted order and there should be a key value, meaning it can not have repeated values.

Non-clustered - A non-clustered index doesn't sort the physical data inside the table. In fact, a non-clustered index is stored at one place and table data is stored in another place. This allows for more than one non-clustered index per table.

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

A deadlock occurs when two or more transactions in a database are waiting on one another to give up locks. An example would be transactionA holds a lock on some rows in a table and needs to update some rows in another table before it can complete. It is an unwanted situation as it brings the whole system to a stop.



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

A livelock occurs when a request for an exclusive lock is repeatedly denied because a series of overlapping shared locks keep interfering. The SQL server will detect the situation after 4 attempts and refuses any further shared locks. A livelock can also occur when read transactions take hold of a table or page, forcing a write transaction to wait indefinitely.

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

	DESCRIPTION	EXAMPLE
capitalize( )	Capitalizes the first letter of a string	s = 'hello' print(s.capitalize( )) Output: Hello
casefold( )	Converts the string to lowercase	s = 'HELLO' print(s.casefold( )) Output: hello
center( )	Returns a new string padded with given character and specified length if given, Syntax: string.center(length[, fillchar])	s = 'hello' print(s.centre(24, '#')) Output: ### hello ###
count( )	An in built function that returns the number of times a substring occurs in the given string	s = "code and code" print(s.count('code')) Output: 2
endswith( )	Returns true if the string ends with given substring, otherwise false	s = "codeandcode" print(s.endswith('code')) Output: True
find( )	This method returns the lowest index or first instance a substring is found in a given string. If not found returns -1	s = "code and code" print(s.find('and')) Output: 5
format( )	Will return a formatted string with the value added in the placeholder position. Syntax: { }.format(value)	txt = "I can run {an:.2f} kilometers!" print(txt.format(an = 8))

		Output: I can run 8.00 kilometers!
index( )	Returns the first position of the substring found.	String ='hellocode' print(string.index('llo') Output: 2
isalnum( )	Checks if all characters in a string are either alphabet or numeric chars. Returns True: If all the characters are alphanumeric False: If one or more characters are not alphanumeric	s = "hello123" print(s.isalnum( )) Output: True s = "hello_123" print(s.isalnum( )) Output: False

isalpha( )	Can be used to check whether all characters in a string is in alphabet	s = "hello" print(s.isalpha( )) Output: True s = "hello_123" print(s.isalpha( )) Output: False
isdigit( )	Returns True if all characters in the string are digits, otherwise returns False	print("123".isdigit( )) Output: True
islower( )	This method checks if all characters in the string are lowercase. Returns true or false.	print('hello'.islower( )) Output: True
isnumeric( )	Returns true if all characters in the string are numeric, otherwise false.	print("1234".isnumeric( )) Output: True
isspace( )	Only returns true if all the characters in the string are whitespace chars, such as ' ', \n, \r	string = "\n\t\n" print(string.isspace( )) Output True

		string = 'code\nfirst\ngirls' print(string.isspace()) Output: False
istitle()	When all words in a string begin with uppercase letters and the remaining characters are lowercase letters, the string is called title-cased. This function ignores digits and special characters	string = "Hello" print(string.istitle()) Output: True s = 'code First Girls' print(s.istitle()) Output: False
isupper()	This method returns whether all characters in a string are uppercase or not.	print((“HELLO”).isupper()) Output: True
join()	An inbuilt method used to join elements of sequence, such as a list, tuple or dictionary, into a string	list = ['h', 'e', 'l', 'l', 'o'] print("".join(list)) Output: hello dict = {'hello': 1, 'coder': 2, 'girls': 3} s = '_'.join(dict) print(s) Output: hello_coder_girls
lower()	Converts all uppercase characters to lowercase and returns the string	s = 'CODE First Girls' print(s.lower()) Output: code first girls
lstrip()	It returns a copy of the string with leading characters removed (based on any argument entered). If no argument is passed, it removes leading spaces.	s = "+-=*codefirstgirls" print(s.lstrip("+-=*")) Output: codefirstgirls
replace()	Used to replace any occurrence of the substring with another given substring.	s = 'Code First Girls' print(s.replace(‘Girls’

	string.replace(old, new, count)	,'Queens')) Output: Code First Queens
rsplit()	Splits the string by specific separator, returns a list of strings	string = "code/first/girls" print(string.rsplit('/')) Output: ['code', 'first', 'girls']
rstrip()	Will return a right trim/ remove trailing chars of the string	string = "clairessss" print(string.rstrip('s')) Output: claire
split()	This method splits a string into a list. You can specify the separator, default separator is any whitespace. rsplit() and split() are very similar, only difference being if the maxsplit argument is set, the rsplit() function splits a string from the right side (from the final character), whereas the split() method splits from the left side (from the first character)	s = "Code, First, Girls" print(s.split(',')) Output: ['Code', 'First', 'Girls']
splitlines()	Used to split a line at a given break. Returns a list of lines in the string broken at the line boundary. Python recognises a number of line boundaries to split including: \n , \r , \r\n , \x1c, \x1d, \x85, \v, \f, & others	s = "Code\nFirst\nGirls\nDegree \nCourse" print(s.splitlines()) Output: ['Code', 'First', 'Girls', 'Degree', 'Course']
startswith()	Returns "True" if a string starts with the given prefix, also has some parameters: Prefix - what to check Start - what position to check from End - end position for the check	s = "Code First Girls" print(s.startswith('Code')) Print(s.startswith('The')) Output: True False
strip()	Returns a string with both the leading and trailing characters removed - can be based on the argument passed	s = " code first girls " print(s.strip()) Output: code first girls

		#removes /n before and after OR; s= “the code first girls” print(s.strip(‘the ‘) Output: code first girls
swapcase()	This method converts all uppercase characters to lowercase and vice versa of the given string.	string = "CODE first GiRls" print(string.swapcase()) Output: code FIRST glrLs

title()	Convert the first character of each word to uppercase (title case), the rest of chars in the string will remain lowercase.	string = "code first girls" print(string.swapcase()) Output: Code First Girls
upper()	All lowercase characters will be converted to uppercase	string = "Claire" print(string.upper()) Output: CLAIRE

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

append()	Adds a single item to the end of the list	list = [‘Code’, ‘First’, ‘Girls’] print(list.append(‘Degree’)) Output: Code First Girls Degree
clear()	Removes all items from the list	list = [‘Code’, ‘First’, ‘Girls’] print(list.clear()) Output [ ]

copy( )	Makes a copy of the list. It is a shallow copy which means if we modify any of the nested list elements, changes are reflected in both the list as they point to the same reference	list = ['Code', 'First', 'Girls'] print('Copied list:' list.copy( )) Output: Copied list: ['Code', 'First', 'Girls']
count( )	Returns the count of the items in the list	list = ['Code', 'First', 'Girls'] print(list.count('Code')) Output: 1
extend( )	Adds all the items of a list (or tuple, string, etc) to the end of the list	list = ['Code', 'First', 'Girls'] list.extend(['Degree', 'Course']) print(list) Output: ['Code', 'First', 'Girls', 'Degree', 'Course']
index( )	Returns the index of the item in the list	list = ['Code', 'First', 'Girls'] print(list.index('First')) Output: 1
insert( )	Inserts an item into the list	list = ['Code', 'Girls'] print(list.insert(1, 'First')) Output: ['Code', 'First', 'Girls']
pop( )	Removes an item from a list that is at a given index	list = ['Code', 'First', 'Girls'] print('pop:' ,list.pop( )) print('list after pop:', list) Output: pop: Girls List after pop: ['Code', 'First']
remove( )	Removes an item from the list, can be a given object	list = ['Code', 'First', 'Girls'] print(list.remove('Code')) Output: ['First', 'Girls']
reverse( )	Reverses the list	list = ['Code', 'First', 'Girls'] print(list.reverse( )) Output: ['Girls', 'First', 'Code']
sort( )	Sorts the list - method sorts the list ascending by default. You can also make a function to decide the	list = ['Code', 'Girls', 'First'] print(list.sort( )) Output: ['Code', 'First', 'Girls']

	sorting criteria(s).	

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

count( )	Counts the number of times a specific value occurs in the tuple	mytuple = (1, 2, 6, 8, 7, 5, 1, 6, 9, 5) x = mytuple.count(5) print(x) Output: 2
index( )	Searches the tuple for a specific value and returns the position it is found, it finds the first occurrence of the specified value. If the value is not found it raises an exception.	mytuple = (1, 2, 6, 8, 7, 5, 1, 6, 9, 5) x = mytuple.index(8) print(x) Output: 3

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

clear( )	Removes all items from the dictionary, returns an empty dictionary	d = {1: "code", 2: "first"} print(d.clear( )) Output : {}
copy( )	Returns a shallow copy of the dictionary (similar to the python list method)	dict = {1: 'Code', 2: 'First',3: 'Girls'} print('Copied dict:' , dict.copy( ))

		Output: Copied dict: {1: 'Code', 2: 'First',3: 'Girls'}
fromkeys()	Creates a dictionary from the given sequence in a new dictionary. Takes two parameters - seq: the sequence to make the dictionary and val: initial values to be assigned to the generated keys (defaults to none)	seq = ('a', 'b', 'c') print(dict.fromkeys(seq, 1)) Output: {'a': 1, 'b': 1, 'c': 1}
get()	Returns the value for the given key if it is found in the dictionary, otherwise returns none. Two parameters key and value.	dict = {1: 'Code', 2: 'First',3: 'Girls'} print(dict.get(1)) Output: Code
items()	Return the list with all dictionary keys with values. Since every dictionary has a key: value pair.	dict = {'A': 'Code', 'B': 'First', 'C': 'Girls'} print(dict.items()) Output: dict_items([('A', 'Code'), ('B', 'First'), ('C', 'Girls')])
keys()	Returns a view object that displays a list of all the keys in the dictionary in order of insertion	dict = {'A': 'Code', 'B': 'First', 'C': 'Girls'} print(dict.keys()) Output: dict_keys(['A', 'B', 'C'])
pop()	Returns and removes the element with the given key. Can accept two parameters - key: the key that's key:value pair will be returned and def - a default value to return if it's not found.	dict = {'A': 'Code', 'B': 'First', 'C': 'Girls'} print(dict.pop('First')) Output: {'A': 'Code', 'C': 'Girls'}
popitem()	Returns and removes the last inserted key-value pair from the dictionary and returns it as a tuple	d = {1: 'Code', 2: 'First', 3: 'Girls'} print(d.popitem()) Output: (3, 'Girls')

setdefault()	Returns the value of a key, if the key is in the dictionary, else it inserts the key with a default value into the dictionary	dict = { "Claire": "Blue", "Sandy": "Red", "Paul": "Yellow"

		} print(dict.setdefault("Paul", "Purple") Output: Yellow

update( )	Updates the dictionary with the elements from another dictionary	dict = {
		"Claire": "Blue",
		"Sandy": "Red",
		"Paul": "Yellow"
		}
		
		print(dict.update({"Ann": "White"})
		
		Output: {
		"Claire": "Blue",
		"Sandy": "Red",
		"Paul": "Yellow",
		"Ann": "White"}

values( )	Returns a list of all the values available in a given dictionary	dict = { 'Code':1, 'First':2, 'Girls':3} print(dict.values( )) Output: dict_values([1,2 , 3])

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

add( )	Adds an element to the set. If the element already exists, the add( ) method does not add the element	set = {"code", "first", "girls"} print(set.add("degree")) Output: {"degree", "code", "first", "girls"}
clear( )	Removes all the elements from the set	set = {"code", "first", "girls"} print(set.clear( ))

		Output: set()
copy()	Returns a copy of the set, same as previous copy methods	set = {"code", "first", "girls"} print(set.copy()) Output:{"code", "first", "girls"}
difference()	Returns a set that contains the difference between two or more sets. The return contains items that exist only in the first set and not from both sets.	set = {"code", "first", "girls"} Set2 = {"code", "degree", "google"} print(set.difference(set2)) Output: {"first", "girls"}
intersection()	Returns a set that contains the similarity between two or more sets, contains only items that exist in both sets. Opposite of difference()	set = {"code", "first", "girls"} Set2 = {"code", "degree", "google"} print(set.intersection(set2)) Output: {"code"}
issubset()	Returns True if all items in set x are present in set y, otherwise returns false	x = {"a", "b", "c"} y = {"f", "e", "d", "c", "b", "a"} print(x.issubset(y)) Output: True
issuperset()	Returns True if all items in the specified set exist in the original set, otherwise it returns False.	x = {"f", "e", "d", "c", "b", "a"} y = {"a", "b", "c"} print(x.issuperset(y)) Output: True
pop()	Removes a random item from the set, it returns the removed item	set = {"code", "first", "girls"} print(set.pop())

		Output: {"code", "girls"}
remove( )	Removes the specified element from the set	set = {"code", "first", "girls"} print(set.remove("first")) Output: {"code", "girls"}
symmetric_diff er ence( )	Returns a set that contains all items from both set, but not the items that are present in both sets.	set = {"code", "first", "girls"} Set2 = {"code", "degree", "google"} print(set.symmetric_difference(set2)) Output: {"first", "degree", "google", "girls"}
union( )	Return a set containing all items from the original set, and all items from the specified set(s). You can specify as many sets you want, separated by commas and if an item is present in more than one set, the result will contain only one appearance of the item	set = {"code", "first", "girls"} Set2 = {"code", "degree", "google"} print(set.union(set2)) Output: {"code", "first", "degree", "google", "girls"}

update( )	Update the set with another set, or any other iterable, If an item is present in both sets, only one appearance of this item will be present in the updated set	set = {"code", "first", "girls"} Set2 = {"code", "degree", "google"} print(set.update(set2)) Output: {"code", "first", "girls", "degree", "google"}

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

read()	Returns the specified number of bytes from the file. Default is -1 which means the whole file. Optional size parameter of the number of bytes to return.	f = open("myfile.txt", "r") print(f.read()) Output: The whole myfile.txt would open
readline()	This method returns one line from the file. You can also specify how many bytes from the line to return, by using the size parameter.	f = open("myfile.txt", "r") print(f.readline()) Output: First line of myfile.txt file
readlines()	Return all lines in the file, as a list where each line is an item in the list object	f = open("myfile.txt", "r") print(f.readlines()) Output: ['The whole myfile.txt would open', 'First line of myfile.txt file']

write()	Writes text to a file. Open the file with "a" for appending, then add some text to the file.	f = open("myfile.txt", "a")
		f.write("Please read me!")
		f.close()
		
		#open and read the file after the appending:
		f = open("myfile.txt", "r")
		print(f.read())
		
		Output: The whole

		myfile.txt would open First line of myfile.txt file Please read me!

writelines()	Writes the items of a list to the file. Open the file with "a" for appending, then add a list of texts to append to the file	f = open("myfile.txt", "a")
		f.writelines(["Thank you for your time."])
		f.close()
		
		#open and read the file after the appending:
		f = open("myfile.txt", "r")
		print(f.read())
		
		Output: The whole myfile.txt would open
		First line of myfile.txt file
		Please read me!
		Thank you for your time.

Source: https://www.geeksforgeeks.org/python-string-join-method/?ref=lbp
https://www.shortcutfoo.com/app/dojos/python-strings/cheatsheet
https://www.w3schools.com/python/python_ref_string.asp