THEORY QUESTIONS ASSIGNMENT

Python based theory

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

Lorelle Brownlee

1. Python theory questions 30 points

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

Python is a computer programming language that is commonly used in software development, data science and other fields. It has a number of benefits that have contributed to its popularity in recent years, including:

- Ease of use with simple syntax
- It is a highly expressive language and doesn't require many lines of code to execute simple tasks.
- It is easy to debug as it is an interpreted language, meaning it is read one line at a time.
- It has high cross-platform compatibility
- It is free and open-source
- It is an object-oriented language, making it easier to reuse code for multiple purposes and helping to reduce the volume of code needing to be written
- It has an extensive standard library, meaning many pre-written functions and modules are available for use.
- It has dynamic memory allocation, and we do not need to manually specify data types of variables on creation.

2. Discuss the difference between Python 2 and Python 3

The syntax in Python 2 is somewhat more complicated than in Python 3, and there are differences in the libraries available for use between the two versions. There is a wider range of libraries available in Python 3 that accommodate its use in a wide range of fields including web development. Python 2 has not officially been in use since 2020. Some commonly noticed differences include:

- Print was a statement in Python 2, but is a function in Python 3
- Strings were stored in ASCII in Python 2, but are stored in UNICODE in Python 3.
- In Python 2, the xrange() function is used for iterations, but in Python 3 the range() function is used instead.

3. What is PEP 8?

PEP 8 stands for Python Enterprise Proposal 8. This is a document that describes guidelines on how to write high-quality, readable Python code. It was officially written in 2001 and aims to improve and maintain readability and consistency of Python code.

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

A computer programme is a set of instructions coded in a programming language that tell a computer to perform a specified task or tasks. A programme is usually stored on a file or disc. It is usually translated by the computer from the programming language it is written in, to computer code which is readable by the computer.

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

A process is the instance of a computer programme that is currently being executed. It is the execution of instructions from a computer programme after it has been loaded from a disk onto memory. It comprises that code for the programme as well as its activity. It may be executed as a single thread, or as many threads simultaneously.

In Windows operating system, all running processes can be seen by looking at Task Manager -> Processes.

6. In computing / computer science what is cache?

Cache is a kind of temporary storage of data, which is usually held in a kind of accessible storage media separate from the computer's main data storage.
It is fast and easy to access, but uses more memory and energy to run than other kinds of storage. Its availability can help speed up access to data that is accessed more frequently. For example, it is often used by web browsers, operating systems and the CPU.

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

A thread is a path followed when a programme is executed. Many programmes run as a single thread, but running as multiple threads is also possible. In some uses such as machine learning, multithread programme execution can help speed up processes. However, careful programming is required to help avoid race conditions and deadlocks, which can cause conflicts. Prevention of this conflicts can be done by using locks to prevent multiple threads from changing the value of the same variable at the same time.

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

Parallelism is any of a number of techniques that can improve programme execution speed by performing multiple computations simultaneously. In order to do this, hardware with multiple processing units is necessary. To enable parallelism, reduction of data dependencies is key. Effectively, tasks are broken into chunks to be run simultaneously.

Concurrency describes running more than one task at the same time. It gives the illusion of parallelism, but actually reflects more than one task being processed at the same time rather than smaller parts of one task as is the case in parallelism. In concurrency, one task isn't fully complete before the next one is started. The aims of concurrency are achieved by context switching.

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

GIL stands for Global Interpreter Lock. This is a lock that allows the Python interpreter, during task execution, to be controlled by only one thread. It can create a bottleneck in multi-threaded code. The advantage of its existence is that it helps prevent deadlocks.

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

- DRY - this stands for ' don't repeat yourself'. It is a guideline in software engineering that aims to reduce code duplication and redundant code, to help optimise performance and readability. It was popularised by the book The Pragmatic Programmer. It states that logic or code that has a specific functionality should ideally only appear once in a programme's code.
- KISS - This stands for 'Keep It Simple Stupid'. It advises that when writing code, you should aim to write only as much code and detail as is required - no less and no more.
- BDUF - This stands for 'Big Design Up Front'. It is generally quoted in app, web development and software development. It advises that when building an application, everything should be complete and working effectively before implementation. It is based on the Waterfall project management methodology. It was popular until more recently when the Agile model became predominant. It relied on the developing team predicting needs upfront. It is not easy to test or to make changes after project completion, so it is less commonly used nowadays.

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

The Python Garbage Collector is a Python feature that helps manage memory. It is designed to fix circular references, which occur when an object in code references itself or two objects reference each other. The Garbage Collector exists as a built in model which can detect circular references and destroy the objects when it is run.

12. How is memory managed in Python?

Memory is managed in Python through a private heap which contains all Python data structures and objects. This private heap is managed internally by the Python memory manager. The Python memory manager has a number of different components, each focusing on different aspects of storage management, for example sharing and cache.

To ensure this private heap has enough memory to accommodate all Python-related data, a raw memory allocator interacts with the memory manager of the operating system. There are also object-specific memory allocators which operate on the heap and manage the distinct memory management requirements of each object type.

Unlike some programming languages, the user has no control over Python memory management.

13. What is a Python module?

A python module is a file that contains statements and definitions. It can be used to define variables, classes and functions in an organised fashion, making it easier to use and understand. Modules can be imported into one another by using the import statement in a Python file, at the top of the script. .

14. What is docstring in Python?

A docstring is short for ‘documentation string’. It is a text description written by a programmer which helps users and contributors understand the wider functionality of the code. It is a string literal, which is used in the definition and description of a class, function, module or method. A docstring can be accessed from the __doc__ attribute or from the help() function.

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

Pickling is a process where a hierarchy of Python objects is converted into a byte or character stream, and ‘unpickling’ is the reversal of this process. Pickling is also known as ‘serialisation’, ‘flattening’ or ‘marshaling’. Often, objects are pickled to enable them to be saved on a disc. Conversion into a byte stream preserves all necessary information in order to be able to

reconstruct the object and use it in another Python script. Unpickling a byte stream can restore the object(s) hierarchy.

Example:
Pickling-

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

Unpickling-

```
unpickle_dict = open("sample_dict..pickle","rb")
sample_dict = pickle.load(unpickle_dict)
print(sample_dict)

#output: {1:"A",2:"B",3:"C",4:"D",5:"E"}
```

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

Some static analysis tools to help find bugs include Pychecker and Pylint.

- Pychecker is an open source static analysis tool that can be used to detect bugs in the source code, and will deliver a warning about the nature of the bug.
- Pylint is also an open source static analysis tool that looks for coding errors. It also checks the length of each line of code, as well as variable names. It controls warnings and errors. It is integratable with popular Python IDEs (integrated developer environments) such as Pycharm and Spyder.

17. How are arguments passed in Python by value or by reference? Give an example.
Pass by value and pass by reference are two kinds of argument passing technique. Pass by value means the function receives a copy of the argument object that has been passed to it by the function call. As a result, the original object stays the same and changes made to it are stored to a copy of it elsewhere.
Pass by reference means the variable is passed to the function directly.

Python's argument passing model is neither - it is Pass by Object Reference. This means that Python functions receive the same object in the memory as referred by the function call. However, it does not receive the variable itself.

Example:

```
def add_one(n):
#n is a name assigned to the function argument when called

#because numbers are immutable,this function

#reassigns n to the number represented by n+1

        n+=1

        return n

y = 5
add_one(a)
#output: 6
#y doesn't change
>>>y
#output: 5
```

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

List comprehension is a sophisticated way to create lists in Python. It consists of four main parts:

- Output expression
- Input sequence
- A variable which represents a member of the input sequence
- A predicate part (optional)

List comprehension example:

```
#syntax for list comprehension is below

#numbers = [expression for variable in iterable if condition == True]

numbers = [num for num in range(10) if  num > 1 and num % 2 == 0]

print(new_list)
```

#output: [2, 4, 6, 8]

Similarly, a dictionary comprehension can define and create a dictionary from a simple expression.

```
# Python code to demonstrate dictionary comprehension
# Keys (fruits) and values (weights) in two lists
keys = ['apple','banana','cherry','dragon fruit','eggplant']
values = [42,106,22,69,140]
# this line is dictionary comprehension
fruit_dict = { k:v for (k,v) in zip(keys, values)}



print (fruit_dict)
#Output : {'apple': 42, 'banana': 106, 'cherry': 22, 'dragon fruit': 69, 'eggplant': 140}
```

19. What is namespace in Python?

A namespace is a system in which every object (such as a method or variable) has a unique name in Python as well as information about the object. The Python dictionary itself is a namespace. Namespaces allow Python to know exactly what method or variable a Python code is referring to. It comes from an amalgamation of the words ‘name’ and ‘space’ referring to the name of the object and the space (referring to scope or information about the object). Namespaces help prevent conflicts in Python.

20.What is pass in Python?

A pass is a statement that is used as a placeholder for code to be written in the future. A programme can be executed with a pass statement present, and when it is run nothing will happen, but it will prevent an error from being raised due to problems with the code. This can help an incomplete programme to run in cases where empty areas of code are incompatible, such as loops and if statements.

21. What is unit test in Python?

A unit test is a method of testing software. Individual units of code are subjected to various tests to determine their functionality and error-liability. It is essentially a code quality assessment. Python comes equipped with a unit test framework, called unittest. Unit testing is usually done

by focusing on ways the code could potentially fail, starting with the smallest unit of code, and working up.

22. In Python what is slicing?

Slicing refers to the use of slice syntax. This is done to select and return a range of characters from a specific object. The start and end indices on the selected range can be stated and the slice() function can then be executed to return the corresponding slice object. Slicing is often useful when, for example, a specific range of items in a list are of interest. Steps can also be used which can, for example, allow a user to select only every other item within a certain range in a list.

23. What is a negative index in Python?

A negative index is the reverse index of an object. For example, the last item in an array will have the index -1, and the second last will be -2, and so on. This can help speed and ease of data retrieval from objects.
An array of 5 items, for example, will have both an index and a negative index. The 5th item in the array will have an index of 4 as well as an index of -1. The use of either can be more beneficial depending on the specific aim.

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

Ternary operators are commonly known as ‘conditional expressions’. These are operators that evaluate whether a condition is true or false. It allows a condition to be tested for being true or false in a single line. It takes its name from the fact that the statement will take three parameters. An example includes:

```
age = input('How old are you?')

if int(age) >= 18:
   ticket_price = 20
else:
   ticket_price = 5

print(f"The entry fee is {ticket_price}")
```

Using a ternary operator:

```
ticket_price = 20 if int(age) >= 18 else 5
print(“The entry fee is {}”.format(ticket_price)
```

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

A variable number of arguments can be passed to a function in Python through the use of special symbols - *args and **kwargs.
*args refer to non-keyword arguments and **kwargs refer to keyword arguments.

Using *args in a function definition allows you to pass a variable number of variable length arguments into a function. It allows the function to take in more arguments than the number of arguments formally defined in the function. The variable associated with the * becomes an iterable, so it becomes possible to iterate over it.

**kwargs allow you to pass a keyworded (meaning a name is provided for the variable as it is passed into the function), variable length argument into a function. A **kwarg acts as a dictionary mapping to each keyword to a value passed alongside it. A **kwarg can be useful if the name of the parameters to a function are not known in advance.

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

In Python, range() and xrange() are both functions used for iterating a specified number of times in for loops. xrange() was removed in Python 3.

- The range() function will return a range object, which is an iterable or a list of integers.
- The xrange() function will return an object called a generator object, which can display numbers only by looping.

The execution of xrange() is faster than range(). range() is slower as it keeps the entire list of items in its memory.

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

Flask is a module in Python that enables easy web application development. It is a kind of web application framework, which is a collection of modules and libraries that enable easy writing of web applications without having to focus on low-level details such as protocol and thread management.

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

A clustered index is an index that sorts the rows of data in a table based on key values. There can only be one clustered index per table, and this index is in a column that determines the sort order of the rows in the table. If a table has no clustered index, the data is stored in an unordered structure, which is called a heap.

A non-clustered index contains key values which have pointers to the storage locations of rows in the heap. Non-clustered indices can help speed up data retrieval during queries.

29. What is a ‘deadlock’ a relational database?

A deadlock is a situation where two or more transactions or tasks are all left waiting indefinitely for the others to give up locks. Locks are in-memory structures designed to preserve the integrity of the data. It is one of the most difficult to manage complications in database management as it can bring the entire system to a standstill. It can be overcome by the database management system detecting the deadlock and stopping one of the involved transactions.

30.What is a ‘livelock’ a relational database?

A livelock is a situation that arises when a request for an exclusive lock is repeatedly denied, as many shared locks overlap and as a result keep interfering with each others processes. Processes keep changing status, preventing completion of tasks. Essentially, the lock keeps moving from process to process without anything moving forward.

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

METHOD	DESCRIPTION	EXAMPLE
capitalize()	This method returns a string with the first letter being a capital letter and the remaining letters being lower case.	restaurant_welcome = "welcome to our restaurant." cap_welcome = restaurant_welcome.capitalize() print (cap_welcome) #output: Welcome to our restaurant.
casefold()	This method returns a string where all letters are lowercase. This method differs from lower() in that it will convert a	restaurant_welcome = "Welcome to our Restaurant. Our name is ß" little_welcome =

	wider variety of characters into lowercase (e.g. special characters).	restaurant_welcome.casefold () print(little_welcome) #output: welcome to our restaurant. our name is ss
center()	This method will put a string in central alignment, and the character used to fill the space between the edge and the centre can be specified in the optional parameter ‘character’. The length of the returned string must be specified in the ‘length’ parameter.	food = "Nut roast" central_food = food.center(25, "ß") print(central_food) #output: ßßßßßßßßNut roastßßßßßßßß
count()	This method returns the number (count) of times a specified value is found in a given string. The start and end positions to be searched within the string can be specified as optional parameters ‘start’ and ‘end’.	restaurant_welcome = "Welcome to our Restaurant. Our name is ß" ur_count = restaurant_welcome.count("ur", 14, 30) print(ur_count) #output: 2
endswith()	This method returns the boolean value True if the specified string ends with a specified value. Otherwise, it will return False. The start and end positions to search within the string can be specified as optional parameters ‘start’ and ‘end’.	restaurant_welcome = "Welcome to our Restaurant. Our name is ß" msg_end = restaurant_welcome.endswith("our", 1, 14) print(msg_end) #output: True
find()	This method finds the first occurrence of a specified value within a given string. It	restaurant_welcome = "Welcome to our Restaurant. Our name is ß"

	will return the start position within the string. If the value is not found, the method will return -1. This is different from the index() method in that index() will instead raise an exception if the value is not found in the string. The start and end positions to search within the string can be specified as optional parameters ‘start’ and ‘end’.	find_ss= restaurant_welcome.find("ss") print(find_ss) #output: -1
format()	This method will reformat the specified value or values and will insert them into the placeholder in the string, which uses curly brackets (‘{}’). This method will return the formatted string. The parameters are the values to be inserted into the string. You will need as many values as there are curly bracket placeholders, and they must be in the same order as they appear in the string.	restaurant_welcome = "Welcome to our restaurant. Our name is {} and we are located in {}.".format('ß', 'London') print(restaurant_welcome) #output: Welcome to our restaurant. Our name is ß and we are located in London.
index()	This method finds the first occurrence of a specified value within a given string. It will return the start position within the string. If the value is not found, the method will return an exception. This is different from the find() method in that find() will instead return -1 if the value is not found in the string. The start and end positions to search within the string can be specified as optional parameters ‘start’ and ‘end’.	restaurant_welcome = "Welcome to our Restaurant. Our name is ß" find_ss= restaurant_welcome.index("s s") print(find_ss) #output: Traceback (most recent call last): File "./prog.py", line 3, in <module> ValueError: substring not found
isalnum()	This method will return the boolean True if all characters	restaurant_welcome = "Welcome2OurRestaurant"

	in the specified string are alphanumeric characters.	chars = restaurant_welcome.isalnum( ) print(chars) #output: True
isalpha()	This method will return the boolean True if all the characters in the specified string are alphabetical characters.	restaurant_welcome = "WelcomeToOurRestaurant" alpha = restaurant_welcome.isalpha() print(alpha) #output: True
isdigit()	This method will return the boolean True if all of the characters are digits. This includes exponent characters. This is different from isnumeric() in that isnumeric() can also check characters such as Unicode fractions and Roman numerals.	restaurant_phone = "01732663104" phone_check = restaurant_phone.isdigit() print(phone_check) #output: True
islower()	This method will return the boolean True if all the alphabetical characters in the string are in the lowercase.	restaurant_welcome = "welcome to our restaurant, our name is ß" check_lower = restaurant_welcome.islower() print(check_lower) #output: True
isnumeric()	This method will return the boolean True if all the characters in the string are numeric. Exponents are considered numeric. This is different from isdigit()	restaurant_phone = "01732663104Ð" check_num = restaurant_phone.isnumeric()

	in that isnumeric() can also check characters such as Unicode fractions and Roman numerals.	print(check_num) #output: True
isspace()	This method will return the boolean True if all characters in the string are white spaces.	sample_text = " _ " x = sample_text.isspace() print(x) #output: False
istitle()	This method will return the boolean True if each word in a string of text starts with a capital letter, with the remaining letters in each word all being lowercase. Symbols, numbers and special characters are ignored.	restaurant_welcome = "Welcome To Our Restaurant." welcome_title = restaurant_welcome.istitle() print(welcome_title) #output: True
isupper()	This method will return the boolean True if all alphabetical characters in the string are capital letters.	restaurant_welcome = "WELCOME!" cap_check = restaurant_welcome.isupper() print(cap_check) #output: True
join()	This method takes all items in a given iterable, joins them together and converts them into a single string, with another string specified as the separator.	diner_orders = ("fish", "chicken", "fish", "nut roast", "pasta", "salad", "beef", "chicken") order_string = "FOOD".join(diner_orders) print(order_string) #output: fishFOODchickenFOODfishFOODnut roastFOODpastaFOODsalad

		FOODbeefFOODchicken
lower()	This method returns a string where all letters are lowercase. This method differs from casefold() in that it has a narrower scope of characters it can convert to lowercase, and often will not convert special characters.	restaurant_welcome = "Welcome to our Restaurant. Our name is ß" little_welcome = restaurant_welcome.lower() print(little_welcome) #output: welcome to our restaurant. our name is ß
lstrip()	This method removes all leading characters. The type of leading characters to be removed can be specified as an optional parameter ‘characters’, but defaults to whitespace.	restaurant_welcome = "xxxxßßßßxxWelcome to our Restaurant. Our name is ß" clean_welcome = restaurant_welcome.lstrip("xß") print(clean_welcome) #output: Welcome to our Restaurant. Our name is ß
replace()	This method replaces a specified string with another specified string. The old and new strings must be specified as parameters, and the number of occurrences of the old string to be replaced can be specified with the optional parameter ‘count’.	restaurant_welcome = "xxxxßßßßxxWelcome to our Restaurant. Our name is ß" B_welcome = restaurant_welcome.replace("x", "ß", 6) print(B_welcome) #output: ßßßßßßßßßßWelcome to our Restaurant. Our name is ß
rsplit()	This method splits a specified string into a list, and does this from right to left.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food.

	The separator can be specified as the optional parameter ‘separator’ but defaults to whitespace. The separator will not be present in the returned list. The number of splits to perform can be specified as the optional parameter ‘maxsplit’, but defaults to -1 meaning all occurrences separated by the separator.	Our name is ß" welcome_list = restaurant_welcome.rsplit(".", 1) print(welcome_list) #output: ['Welcome to our Restaurant. We hope you enjoy our food', ' Our name is ß']
rstrip()	This method removes any lagging or trailing characters at the end of a string. The characters to be removed can be specified as an optional parameter ‘characters’, otherwise it will default to whitespace.	restaurant_welcome = "Welcome to our Restaurant. Our name is ßßßßßxx...." tidy_welcome = restaurant_welcome.rstrip("x.") print(tidy_welcome) #output: Welcome to our Restaurant. Our name is ßßßßß
split()	This method splits a specified string into a list. The separator can be specified as the optional parameter ‘separator’ but defaults to whitespace. The separator will not be present in the returned list. The number of splits to perform can be specified as the optional parameter ‘maxsplit’, but defaults to -1 meaning all occurrences separated by the separator.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food. Our name is ß" welcome_list = restaurant_welcome.split(".", 1) print(welcome_list) #output: ['Welcome to our Restaurant', ' We hope you enjoy our food. Our name is ß']
splitlines()	This method splits a string into a list. The splitting occurs	restaurant_welcome = "Welcome to our

	at line breaks. The line breaks can be included in the returned output if the optional parameter ‘keeplinebreaks’ is set to True. It defaults to False.	Restaurant.\n We hope you enjoy our food.\n Our name is ß" welcome_lines = restaurant_welcome.splitlines () print(welcome_lines) # output: ['Welcome to our Restaurant.', ' We hope you enjoy our food.', ' Our name is ß']
startswith()	This method will return the boolean True if a specified string starts with a specified value. The start and end positions to search within the string can be specified as optional parameters ‘start’ and ‘end’.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food. Our name is ß" start_check = restaurant_welcome.startswith("Restaur", 15, 25) print(start_check) #output: True
strip()	This method removes any leading characters at the beginning of a string. The characters to be removed can be specified as an optional parameter ‘characters’, otherwise it will default to whitespace.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food. Our name is ß" new_welcome = restaurant_welcome.strip("Welcome turRsan") print(new_welcome) #output: . We hope you enjoy our food. Our name is ß
swapcase()	This method will return a string where all capital letters from the original string have been replaced with lowercase letters and vice versa.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food. Our name is ß"

		inverse_welcome = restaurant_welcome.swapcase() print(inverse_welcome) #wELCOME TO OUR rESTAURANT. wE HOPE YOU ENJOY OUR FOOD. oUR NAME IS SS
title()	This method will return a string where the first letter in every word from the original string is a capital letter. If a letter after the first letter in a word in the original string is already capitalised, this method will convert it to lowercase.	restaurant_welcome = "Welcome to our RESTAURANT. We hope you enjoy our food. Our name is ß" welcome_title = restaurant_welcome.title() print(welcome_title) #output: Welcome To Our Restaurant. We Hope You Enjoy Our Food. Our Name Is Ss
upper()	This method returns a string where all characters in the original string have been converted to capital letters.	restaurant_welcome = "Welcome to our Restaurant. We hope you enjoy our food. Our name is ß" cap_welcome = restaurant_welcome.upper() print(cap_welcome) #output: WELCOME TO OUR RESTAURANT. WE HOPE YOU ENJOY OUR FOOD. OUR NAME IS SS

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

METHOD	DESCRIPTION	EXAMPLE
append()	This method adds (or “appends”) an element to the end of a list.	mains = ["fish", "chicken", "nut roast"] sides = ["broccoli", "potatoes", "beans"] mains.append(sides) print(mains) #Output: ['fish', 'chicken', 'nut roast', ['broccoli', 'potatoes', 'beans']]
clear()	This method removes all elements from a list.	mains = ["fish", "chicken", "nut roast"] mains.clear() print(mains) #Output: []
copy()	This method returns a copy of the list specified.	mains = ["fish", "chicken", "nut roast"] mains_duplicate = mains.copy() print(mains_duplicate) #output : [‘fish’, ‘chicken’, ‘nut roast’]
count()	This method returns the number (or count) of the elements with the value specified.	mains = ["fish", "chicken", “fish”, "nut roast"] fish_count = mains.count("fish") print(fish_count) #output: 2
extend()	This method adds specific	mains = ["fish", "chicken",

	elements of a list or any iterable to the end of a selected list. This differs from append() in that it adds the elements as individual elements, rather than adding a whole list as a single element in the case of appending a list to a list.	"nut roast"] sides = ["broccoli", "potatoes", "beans"] mains.extend(sides) print(mains) #output: ['fish', 'chicken', 'nut roast', 'broccoli', 'potatoes', 'beans']
index()	This method returns the position of the first occurrence within a list of the value specified as the parameter.	mains = ["fish", "chicken", "nut roast"] dish_index = mains.index("nut roast") print(dish_index) #output: 2
insert()	This method inserts the value specified into the position specified in the selected list.	mains = ["fish", "chicken", "nut roast"] mains.insert(1, "beef") print(mains) #output: ['fish', 'beef', 'chicken', 'nut roast']
pop()	This method removes the list element at the position specified.	mains = ["fish", "chicken", "nut roast"] mains.pop(2) print(mains) #output: ['fish', 'chicken']

remove()	This method removes the first occurrence of the element with the value specified within a selected list.	mains = ["fish", "chicken", "fish", "nut roast"] mains.remove("fish") print(mains) #output: ['chicken', 'fish', 'nut roast']
reverse()	This method reverses the sort order of the elements in a selected list.	mains = ["fish", "chicken", "nut roast"] mains.reverse() print(mains) #output: ['nut roast', 'chicken', 'fish']
sort()	This method sorts the order of elements in a list. The default order is ascending, but the sort order criteria can be amended. In the case of alphabetical strings, the elements will be sorted by alphabetical order.	mains = ["fish", "chicken", "nut roast"] mains.sort() print(mains) #output: ['chicken', 'fish', 'nut roast']

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

METHOD	DESCRIPTION	EXAMPLE
count()	This method, when used on a tuple, returns the number of times a specific value appears in the selected tuple.	diner_orders = ("fish", "chicken", "fish", "nut roast", "pasta", "salad", "beef", "chicken")

		chicken_count = diner_orders.count("chicken") print(chicken_count) #output: 2
index()	This method, when used on a tuple, identifies the first occurrence within the tuple of the value specified. If the value is not found in the tuple, an exception is raised.	diner_orders = ("fish", "chicken", "fish", "nut roast", "pasta", "salad", "beef", "chicken") carbonara = diner_orders.index("pasta") print(carbonara) #output: 4

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

METHOD	DESCRIPTION	EXAMPLE

clear()	This method removes all elements from the selected dictionary.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		menu_choices.clear()
		
		print(menu_choices)
		
		#output: {}

copy()	This method returns a copy of the dictionary specified.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}

		duplicate_menu = menu_choices.copy() print(duplicate_menu) #output: {'Tom': 'Nut roast', 'Sarah': 'Chicken', 'Stephen': 'Beef', 'Kevin': 'Salmon'}
fromkeys()	This method returns a new dictionary with the keys and values specified.	mains = ("fish", "chicken", "nut roast") sides = ("broccoli", "potatoes", "beans") full_menu = dict.fromkeys(mains, sides) print(full_menu) #output: {'fish': ('broccoli', 'potatoes', 'beans'), 'chicken': ('broccoli', 'potatoes', 'beans'), '

get()	This model will return the value matched with a specified key.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		dinner = menu_choices.get("Tom")
		
		print(dinner)
		
		#output: Nut roast

items()	This method returns a view object containing the key-value pairs of the dictionary, presented as tuples in a list.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		choices_by_person = menu_choices.items()
		
		print(choices_by_person)
		
		#output: dict_items([('Tom', 'Nut roast'), ('Sarah', 'Chicken'), ('Stephen', 'Beef'), ('Kevin', 'Salmon')])

keys()	This method returns a view object that contains the keys of the specified dictionary, but in the form of a list.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		diners = menu_choices.keys()
		
		print(diners)
		
		#output: dict_keys(['Tom', 'Sarah', 'Stephen', 'Kevin'])

pop()	This method removes a specified item from a specified dictionary. The method will return the value of the removed item.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		updated_choices = menu_choices.pop("Stephen")
		
		print(updated_choices)

		#output: Beef

popitem()	This method removes the item that was most recently inserted into the dictionary.
	In previous versions of Python (<v 3.7), this method would remove a random item instead.
	The return value of the method is the removed item, in the form of a tuple.
		menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		last_choice =menu_choices.popitem()
		
		print(last_choice)
		
		#output: (‘Kevin’, ‘Salmon’)

setdefault()	This method will return the value of the item with the key specified. An optional value can be specified as a parameter, so if a value does not already exist for this key, the value in the parameter field will be inserted and the value will be returned. If the key does not already exist, no value will be returned.	menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		lunch_choices = menu_choices.setdefault("Patricia", "Pasta bake")
		
		print(lunch_choices)
		
		#output: Pasta bake

update()	This method inserts specified items into the specified dictionary.
	The inserted items can be a dictionary, or any iterable object with key-value pairs.
		menu_choices = {
		"Tom": "Nut roast",
		"Sarah": "Chicken",
		"Stephen": "Beef",
		"Kevin": "Salmon"
		}
		
		menu_choices.update({"Patricia": "Pasta bake"})
		
		print(menu_choices)
		
		#output: {'Tom': 'Nut roast',

		'Sarah': 'Chicken', 'Stephen': 'Beef', 'Kevin': 'Salmon', 'Patricia': 'Pasta bake'}
values()	This method will return a view object containing the values of a dictionary, in the form of a list. It will also get updated when values are changed in the dictionary.	menu_choices = { "Tom": "Nut roast", "Sarah": "Chicken", "Stephen": "Beef", "Kevin": "Salmon" } meals = menu_choices.values() menu_choices["Kevin"] = "Cod" print(meals) #output: dict_values(['Nut roast', 'Chicken', 'Beef', 'Cod'])

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

METHOD	DESCRIPTION	EXAMPLE
add()	This method adds an element to an existing set. If the element is already present in the set, this method will not add it. It will return elements in no particular order.	diner_orders = {"fish", "chicken", "nut roast", "pasta", "salad", "beef"} diner_orders.add("ham") print(diner_orders) #output: {'ham', 'salad', 'chicken', 'pasta', 'fish', 'beef', 'nut roast'}
clear()	This removes all elements in a selected set.	diner_orders = {"fish", "chicken", "nut roast",

		"pasta", "salad", "beef"} diner_orders.clear() print(diner_orders) #output: set()
copy()	This method copies a set.	diner_orders = {"fish", "chicken", "nut roast", "pasta", "salad", "beef"} duplicate_orders = diner_orders.copy() print(duplicate_orders) #output: {'chicken', 'pasta', 'salad', 'nut roast', 'beef', 'fish'}
difference()	This method returns a set comprising the difference between two sets being compared against each other. It's similar to subtraction. If all items in the first set exist in the second set, the method will return an empty set.	menu_1 = {"fish", "chicken", "fish", "nut roast"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_2.difference(menu_1) print(final_menu) #output: {'broccoli', 'potatoes', 'beans'}
intersection()	This method returns the set of elements that exist between two or more sets.	menu_1 = {"fish", "chicken", "fish", "nut roast"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_1.intersection(menu_2) print(final_menu)

		#output: {'nut roast', 'chicken', 'fish'}
issubset()	This method returns a Boolean. It will return True if all items in the first set exist in the set detailed in the parameter, otherwise it will return False.	menu_1 = {"fish", "chicken", "fish", "nut roast"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_1.issubset(menu_2) print(final_menu) #output: True
issuperset()	This method returns a Boolean. It will return true if all items in the set detailed in the parameter exist in the first set, otherwise it will return False.	menu_1 = {"fish", "chicken", "fish", "nut roast"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_1.issuperset(menu_2) print(final_menu) #output: False
pop()	This method removes a single random item from a selected set.	menu = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} menu.pop() print(menu) #output: {'broccoli', 'fish', 'beans', 'potatoes', 'chicken'}
remove()	This method will remove a	menu = {"fish", "chicken", "nut

	specified element from a specified set. If the specified item does not exist, it will raise an error (unlike the discard() method)	roast", "broccoli", "potatoes", "beans"} menu.remove("nut roast") print(menu) #output: {'fish', 'chicken', 'potatoes', 'broccoli', 'beans'}
symmetric_differ ence()	This method will return all items that are not present in both sets (each item in the returned set will have only been present in one of each of the selected sets).	menu_1 = {"fish", "chicken", "fish", "nut roast", "hot dog"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_1.symmetric_difference(menu_2) print(final_menu) #output: {'broccoli', 'hot dog', 'beans', 'potatoes'}
union()	This method returns a set that contains all items from both the first set and the set specified in the parameter. This method can be used on more than one set, or on any iterable.	menu_1 = {"fish", "chicken", "fish", "nut roast", "hot dog"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"} final_menu = menu_1.union(menu_2) print(final_menu) #output: {'nut roast', 'fish', 'potatoes', 'chicken', 'hot dog', 'beans', 'broccoli'}
update()	This method updates the original set through adding items from the set selected as the parameter. If an item is present in both sets, it will not	menu_1 = {"fish", "chicken", "fish", "nut roast", "hot dog"} menu_2 = {"fish", "chicken", "nut roast", "broccoli", "potatoes", "beans"}

	be added to the original set.	#output: {'potatoes', 'hot dog', 'chicken', 'nut roast', 'broccoli', 'beans', 'fish'}

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

METHOD	DESCRIPTION	EXAMPLE
read()	This method returns the contents of a file to the specified number of bytes. If the number of bytes is not specified (using the 'size' parameter), it will default to -1, meaning the whole file is read.	f = open("Restaurantwelcome.txt", "r") print(f.read(40)) #output: Welcome to our Restaurant. We hope you e
readline()	This method returns one line from the file. The number of bytes from the line to be returned can be specified in the optional size parameter. If you want to read more than one line, repeating the method will read the next line down.	Contents of file.txt: Thank you for trying readline. We know you'll have fun. f = open("demofile.txt", "r") print(f.readline(5)) print(f.readline(5)) #output: Thank We kn
readlines()	This method returns a list that contains each line in the file as a separate list item. The number of bytes returned can be specified by using the optional 'hint' parameter, which determines the number of bytes retrieved.	f = open("Restaurantwelcome.txt", "r") print(f.readlines(150)) #output: ['Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ\n', 'Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ\n', 'Welcome to our Restaurant. We hope you enjoy our food. Our name is

		ÃŸ\n']
write()	This method writes to an existing file. To write to a file, the file must be opened with the open() method. The way of writing to a file can be determined by the parameter added to the open() function - either “a” (append) or “w” (write). W will write over the current contents of the file and a will add to the end of the current contents of the file.	f = open("Restaurantwelcome.txt", "a") f.write("\nThank you for coming to our restaurant.") f.close() f = open("Restaurantwelcome.txt", "r") print(f.read()) f.close() #output: Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ Thank you for coming to our restaurant.
writelines()	This method writes the items of a list to a specified file. The list of strings or byte objects that will be added to the file are specified in the parameter ‘list’. The file must be opened with the open() method, and the “a” parameter in open() will add the items to the current position in the file (defaulting to the end). The “w” parameter in open() will mean the new list objects will write over the current contents of the file.	f = open("Restaurantwelcome.txt", "a") f.writelines(["Thank you for coming to our restaurant.", "Come again sometime."]) f.close() f = open("Restaurantwelcome.txt", "r") print(f.read()) f.close() #output: Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ Welcome to our Restaurant. We hope you enjoy our food. Our name is ÃŸ Thank you for coming to our restaurant.Thank you for

		coming to our restaurant.Come again sometime.