PAULA MANESE
CFG-NANODEGREE
SUBMITTED TO: MR. SCOTT
JUNE 25, 2022

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?

ANSWER:

-Python is an interpreted, object-oriented, high-level programming language with dynamic semantics.

-Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance.

Python's Main Features:

-Easy to learn and use
-Expressive language
-Interpreted language
-Large Standard Library
-Cross-platfrom language
-Free and Open Source
-Object-Oriented language
-GUI programming support
-Extensible
-Intergrated
-Embedable

2 . Discuss the difference between Python 2 and Python 3

ANSWER:

-Python 2 is a python programming version that has been released with several new feature updates and fixes to the earlier python versions of 0.x and 1.x. The first version in python 2.x is Python 2.0 which was released on 16 October 2000. The latest version is Python 2.7, and it was discontinued from the year 2020 and there will be no more releases in Python 2.x.

-Python 3 is an updated version of python programming that has been released mainly to fix the fundamental design problems that exist in Python 2. The changes required to fix the flaws could not be implemented while retaining full backward compatibility with the 2.x series, which necessitated a new major version number which is 3.x.

THE DIFFERENCE:

Basis of comparison	Phython 3	Phython 2
Release Date	2008	2000
Function print	print (“hello”)	print “hello”
Division of Integers	Whenever two integers are divided, you get a float value	When two integers are divided, you always provide integer value.
Unicode	In Python 3, default storing of strings is Unicode.	To store Unicode string value, you require to define them with “u”.
Syntax	The syntax is simpler and easily understandable.	The syntax of Python 2 was comparatively difficult to understand.
Rules of ordering Comparisons	In this version, Rules of ordering comparisons have been simplified.	Rules of ordering comparison are very complex.
Iteration	The new Range() function introduced to perform iterations.	In Python 2, the xrange() is used for iterations.
Exceptions	It should be enclosed in parenthesis.	It should be enclosed in notations.
Leak of variables	The value of variables never changes.	The value of the global variable will change while using it inside for-loop.
Backward compatibility	Not difficult to port python 2 to python 3 but it is never reliable.	Python version 3 is not backwardly compatible with Python 2.
Library	Many recent developers are creating libraries which you can only use with Python 3.	Many older libraries created for Python 2 is not forward-compatible.

3. What is PEP 8?

ANSWER:

-is a tool to check your Python code against some of the style conventions.

-by following the rules of PEP-8 you make your code more readable and understandable resulting in you spending less time trying to understand why it was written in the first place.

Few important points:
Indent using spaces, not tabs
Indent using 4 spaces.
Python files are encoded in UTF-8
Use maximum 80 columns for your code
Write each statement on its own line
Functions, variable names and file names are lowercase, with underscores between words (snake_case)
Functions, variable names and file names are lowercase, with underscores between words (snake_case)
Package names are lowercase and do not have underscores between words
Variables that should not change (constants) are written in uppercase
Variable names should be meaningful
Add useful comments, but avoid obvious comments
Add spaces around operators
Do not use unnecessary whitespace
Add a blank line before a function
Add a blank line between methods in a class
Inside functions/methods, blank lines can be used to separate related blocks of code to help readability

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

ANSWER:

- is a sequence of instructions in a programming language that a computer can execute or Interpret.
- is also alternatively referred to as a software application. The computer programs are generally written using any human readable high level programming language.

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

ANSWER:

- is an instance of a program running in a computer. It is close in meaning to task, a term used in some operating systems.
- depending on the operating system (OS), a process maybe made up of multiple threads of execution that executes instructions concurrently.

6. In computing/computer science what is a cache?

ANSWER:

- is a hardware or software component that stores data so that future requests for that data can be served faster; the data stored in a cache might be the result of an earlier computation or a copy of data stored elsewhere.
- computers use caching to store data that is likely to be used soon, in a faster form of memory, so that it can be accessed more easily in the future.
- is a reserved storage location that is used to speed up the loading of websites, browsers, and apps by collecting temporary data. There are many types of caching available, whether it's on a computer, laptop, phone, web browser, or app. Data can be retrieved quickly in a cache, which in turn speeds up the process of running a device.

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

ANSWER:

THREAD

- is a small set of instructions designed to be scheduled and executed by the CPU independently of the parent process. For example, a program may have an open thread waiting for a specific event to occur ot running a separate job, allowing the main program to perform other tasks. A program is capable of having multiple threads open at once and terminates or suspends them after the task is completed or the program is closed.
    - There are two types of thread:
        - User-level thread
        - Kernel-level thread

MULTITHREADING

- a feature of a computer operating system that allows it to run several parts of a program concurrently or in a quick succession.

MULTITHREADING MODELS

-Many-to-One Model

-One-to-One Model

-Many-to-many Model

BENEFITS OF MULTITHREADING

- Resource sharing
- Utilization of MultipleProcessor Architecture
- Reduced Context Switching Time
- Economical

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

ANSWER:

CONCURRENCY

- relates to an application that is processing more than one task at the same time.
- is an approach that is used for decreasing the response time of the system by using the the single processing unit.
- is achieved through the interleaving operation of processes on the CPU or in other words by the context switching.

PARALLELISM

- is related to an application where tasks are divided into smaller sub-tasks that are processed seemingly simultaneously or parallel. It is used to increased the throughput and computational speed of the system by using multiple processors. It enables single sequential CPUs to do lot of things “seemingly” simultaneously.

DIFFERENCE BETWEEN CONCURRENCY AND PARALLELISM

Concurrency is when two tasks can start, run, and complete in overlapping time periods. Parallelism is when tasks literally run at the same time, eg. on a multi-core processor.
Concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations.
Concurrency is about dealing with lots of things at once. Parallelism is about doing lots of things at once.
An application can be concurrent but not parallel, which means that it processes more than one task at the same time, but no two tasks are executing at the same time instant.
An application can be parallel but not concurrent, which means that it processes multiple sub-tasks of a task in a multi-core CPU at the same time.
An application can be neither parallel nor concurrent, which means that it processes all tasks one at a time, sequentially.
An application can be both parallel and concurrent, which means that it processes multiple tasks concurrently in a multi-core CPU at the same time.

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

ANSWER:

- is a mutex that allows only one thread at a time to have the control of the Python interpreter. In other words, the lock ensures that only one thread is running at any given time. Therefore, it is impossible to take advantage of multiple processors with threads.

HOW DOES GIL WORK?

- Unlike the other programming languages, Python has a “reference-counter” for memory management. When an object is declared in python, there’s a reference-counter variable dedicated to it. This will keep track of the number of references that point to the particular object.
- The GIL is simple to implement and was easily added to Python. It provides a performance increase to single-threaded programs as only one lock needs to be managed.

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

ANSWER:

KISS

- KEEP IT SIMPLE STUPID
- the KISS principle states that most systems work best if they are kept simple rather than made complicated; therefore, simplicity should be a key goal in design, and unnecessary complexity should be avoided.

DRY

DON’T REPEAT YOURSELF

- aimed at reducing repetition of software patterns.
- the DRY principle is stated as "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system".

BDUF

BIG DESIGN UPFRONT

- also known as the art of doing what shouldn’t have been done in the first place.
- is indicative that the design was completed and perfected even before it was implemented.

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

ANSWER:

- deletes unwanted objects (built-in types or class instances) automatically to free the memory space. The process by which Python periodically frees and reclaims blocks of memory that no longer are in use.
- runs during program execution and is triggered when an object's reference count reaches zero. An object's reference count changes as the number of aliases that point to it changes.
- is implemented in Python in two ways: reference counting and generational. When the reference count of an object reaches 0, reference counting garbage collection algorithm cleans up the object immediately. If you have a cycle, 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?

ANSWER:

- Memory management in simple terms means , the process of providing memory required for your program to store data and freeing up unused data in memory is called memory management.
- Providing memory is called memory allocation. Freeing up memory is called memory de-allocation.
- In Python , Memory manager is responsible for allocating and de-allocating memory.

How memory is managed in Python

-Python has a private heap that stores our program's objects and data structures. Python memory manager takes care of the bulk of the memory management work and allows us to concentrate on our code.

-Types of Memory Allocation:

-Static memory

-Dynamic memory

-Python Garbage Collection

13. What is a Python Module?

ANSWER:

- help us to organize and group the content by using files and folders.
- we can consider a module to be the same as a code library or a file that contains a set of functions that you want to include in your application.
- With the help of modules, we can organize related functions, classes, or any code block in the same file. So, It is considered a best practice while writing bigger codes for production-level projects in Data Science is to split the large Python code blocks into modules containing up to 300–400 lines of code.

The module contains the following components:

- Definitions and implementation of classes,
- Variables, and
- Functions that can be used inside another program

-Advantages of Modules

- Reusability
- Simplicity
- Scoping

14. What is docstring in Python?

ANSWER:

- is a string used to document a Python module, class, function or method, so programmers can understand what it does without having to read the details of the implementation.
- We can also use triple """ quotations to create docstrings.

Example:

```
"""Takes in two numbers, returns their product."""
```

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

ANSWER:

- The pickle module is used for implementing binary protocols for serializing and de-serializing a Python object structure.

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

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

* Only after importing pickle module we can do pickling and unpickling. Importing pickle can be done using the following command —--- import pickle

Pickle examples:

-Below is a simple program on how to pickle a list.

-Pickle a simple list: Pickle_list1.py

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

- In the above code, list – “mylist” contains four elements (‘a’, ‘b’, ‘c’, ‘d’). We open the file in “wb” mode instead of “w” as all the operations are done using bytes in the current working directory. A new file named “datafile.txt” is created, which converts the mylist data in the byte stream.

Unpickle a simple list: unpickle_list1.py

```
import pickle
pickle_off = open(“datafile.txt”, “rb”)
emp = pickle.load(pickle_off)
print(emp)
```

- Output: on running above scripts, you can see your mylist data again as output.

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

ANSWER:

- Pychecker and Pylint are the static analysis tools that help to find bugs in python.

- Pychecker is an opensource tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.

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

- Pychecker can be simply installed by using pip package pip install Pychecker if suppose if you use python 3.6 version use upgrade pip install Pychecker --upgrade Pylint can be simply installed by using pip package.

    pip install Pylint

    if suppose if you use python 3.6 version use upgrade

    pip install Pylint --upgrade

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

ANSWER:

- Python uses a mechanism, which is known as "Call-by-Object", sometimes also called "Call by Object Reference" or "Call by Sharing”
- If you pass immutable arguments like integers, strings or tuples to a function, the passing acts like Call-by-value. It's different, if we pass mutable arguments.
- All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

EXAMPLE:

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

OUTPUT:

Inside the function {'Archana': 28, 'krishna': 25, 'Ramesh': 32, 'vineeth': 25, 'alok': 30, 'Nevadan': 28}

Outside the function: {'Archana': 28, 'krishna': 25, 'Ramesh': 32, 'vineeth': 25, 'alok': 30, 'Nevadan': 28}

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

ANSWER:

DICTIONARY COMPREHENSIONS

- A dictionary comprehension is similar to a list comprehension except it returns a dictionary of keys and values instead of a plain list.

SYNTAX OF A DICTIONARY COMPREHENSION
{key: value for iterate_list_or_dict if conditional}

- As you can see the syntax for a Dictionary Comprehension is very similar to that of a List Comprehension. However, a Dictionary Comprehension is surrounded by {} (curly braces). Python sees the curly braces and the syntax inside and is able to determine that this is a Dictionary Comprehension.

- The first part is where we define the key and value combination to return to the new dictionary for the current item being processed. We always place our key first, then a :and then the value.

- We then iterate in the second part. We can iterate through the items in an array, or we can iterate through the keys and values of another dictionary.

- Lastly we have our conditional statement again which we can use to filter out any data items from our final dictionary based on an item's key or value.

EXAMPLES:

Example 1: Create a dictionary of numbers and their squared values
- a dictionary of numbers and their corresponding squared values. In this case our numbers will be the key and the squared values will be the value. We will loop through a sequence from 1 to 10 to create this dictionary.

This will be our dictionary comprehension:

```
numbers_with_squares = {x: x*x for x in range(1, 11)}
print numbers_with_squares
# Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Example 2: Trim white-space from a dictionary with string values

- dictionary containing string details about a person. To make sure our data is neat, we want to go through each item in the dictionary and delete any leading or trailing white-space from each value. To loop through the dictionary we will use the dictionary's iteritems() method which lets us loop through both the keys and values. We will also use Python's string's strip() method to eliminate any leading or trailing white-space.

Our dictionary comprehension should look something like this:

```
details = {
                    "name": "Ivan    ",
                "surname": "    Kahl",
        "email": "  ivan@example.com     "
    }
details_stripped = {key: value.strip() for key, value in details.iteritems()}
print details_stripped
# Output: {'surname': 'Kahl', 'name': 'Ivan', 'email': 'ivan@example.com'}
```

Example 3: Remove any items from a dictionary where the string value is empty

- iterate through a dictionary using a dictionary comprehension and remove any items in an existing dictionary that are empty/blank. To do this, we need to have a conditional expression that will return True if the value is not empty.

Our dictionary comprehension should look something like this:

```
data = {
        "song_name": "Hey There Delilah",
         "artist": "Plain White T's",
         "key": "D",
        "strumming_pattern": "",
        "timing": ""
    }
nonempty_data = {key: value for key, value in data.iteritems() if value != ""}

print nonempty_data
# Output: {'song_name': 'Hey There Delilah', 'key': 'D', 'artist': "Plain White T's"}
```

-As you can see our comprehension successfully returned only the items that had a non-empty value.

LIST COMPREHENSIONS

- A list comprehension processes data to return a list of values afterwards.

SYNTAX OF A LIST COMPREHENSION

```
[some_function(x) for x in original_list if certain_condition]
```

- The first thing to notice is that because this is a list comprehension, it is surrounded in [](square brackets). Python recognises a list and then, by looking at the syntax inside the square brackets, is able to determine that this is a list comprehension.

The first thing that we put inside these square brackets is what we want to return after processing each data item (represented by `x` in this case). In this case, we are performing `some_function` on `x` and then returning it.

Next, we have our for loop where we loop through each item in another iterable (this could be an existing list or a sequence generated with the `range` function). Here we loop through `original_list` and assign each item in `original_list` to `x`, which we are processing in the first part of the comprehension. This variable can be called anything though, it doesn't have to be `x`.

The last part of the list comprehension is completely optional. It allows us to filter values based on a certain condition. This can be any conditional expression and can include `x`(the current data item you are processing). If the condition evaluates to `True`, the data item is processed and inserted into the new list which is returned afterwards, otherwise it is skipped.

This whole list comprehension returns a new list containing the new values that have been processed.

List Comprehension Examples

Example 1: Square all numbers in a list

- let's just square all the numbers in a list using a list comprehension. So we will be processing each original number (represented by `x` in this case) by squaring it (`x * x`). No conditional statement is necessary because we are simply squaring all the numbers regardless of any conditions.

Our comprehension would look as follows:

```
n = [1, 6, 4, 2, 9, 15, 5]
squared_n = [x * x for x in n]

print squared_n
# Output: [1, 36, 16, 4, 81, 225, 25]
```

- As you can see, it produced a new list where each item was the square of the original item in the original list `n`.

Example 2: Return only even numbers

- In this next example, we will be given a list of numbers. We must go through and filter this list so that we only return the even numbers in the original list. In this case, no processing is required because we want the original data item in the returned list. We will however need a conditional statement that will return True when our data item (`x`) is even (i.e. `x % 2 == 0`).

Our list comprehension will look as follows:

```
n = [1, 7, 4, 5, 8, 12, 11]
even_n = [x for x in x if x % 2 == 0]

print even_n
# Output: [4, 8, 12]
```

- As you can see, the list comprehension only returned the even numbers in the new list.

Example 3: Return all the odd numbers doubled

- Now we have to find all the odd numbers in the list and double them before returning them in a new list. This means we will have to filter out all the even numbers using a conditional (`x % 2 != 0`) and then double the remaining numbers (`2 * x`).

Here is our list comprehension:

```
n = [4, 7, 1, 3, 2, 10]
doubled_odd_n = [2 * x for x in n if x % 2 != 0]

print doubled_odd_n
# Output: [14, 2, 6]
```

- You can see how it removed all the even numbers and doubled the remaining odd numbers.

Example 4: Double odd numbers and half even numbers

- For a bit more of a challenging example, let's loop through all the numbers and double the odd numbers and half the even numbers. First thing to notice is that we aren't filtering any numbers out so a conditional for the list comprehension is not necessary. Our processing will be handling the doubling and halving of values based on whether they are odd or even. To do this we can use an inline if statement. This means our processing part will look as follows:

  `x / 2 if x % 2 == 0 else 2 * x`

  This inline if statement will return the number halved if the number is even, otherwise it will return the double of the number.

This means our list comprehension will look as like:

```
n = [1, 7, 3, 4, 10, 9]
manipulated_n = [x / 2 if x % 2 == 0 else 2 * x for x in n]

print manipulated_n
# Output: [2, 14, 6, 2, 5, 18
```

- As you can see our comprehensions able to double the odd numbers and half the even ones successfully.

19. What is namespace in Python?

ANSWER:

- namespace is basically a system to make sure that all the names in a program are unique and can be used without any conflict. You might already know that everything in Python—like strings, lists, functions, etc.—is an object. Another interesting fact is that Python implements namespaces as dictionaries. There is a name-to-object mapping, with the names as keys and the objects as values. Multiple namespaces can use the same name and map it to a different object.

Here are a few examples of namespaces:

- Local Namespace: This namespace includes local names inside a function. This namespace is created when a function is called, and it only lasts until the function returns.
- Global Namespace: This namespace includes names from various imported modules that you are using in a project. It is created when the module is included in the project, and it lasts until the script ends.
- Built-in Namespace: This namespace includes built-in functions and built-in exception names.

20. What is that pass in Python?

ANSWER:

- a pass statement is a null statement. But the difference between pass and comment is that comment is ignored by the interpreter whereas pass is not ignored.
- is generally used as a placeholder i.e. when the user does not know what code to write. So user simply places pass at that line. Sometimes, pass is used when the user doesn't want any code to execute. So user can simply place pass where empty code is not allowed, like in loops, function definitions, class definitions, or in if statements. So using pass statement user avoids this error.

SYNTAX:

```
pass
```

EXAMPLE:

```
class Person:
    pass
```

21. What is that unit test in Python?

ANSWER:

- Unit Testing is the first level of software testing where the smallest testable parts of a software are tested. This is used to validate that each unit of the software performs as designed.
- Python unittests can be used to test single functions, entire modules or a complete interface. In this article, we will write tests on a simple function using a common testing framework called unittest.

22. In Python what is slicing?

ANSWER:

- In Python, slicing makes it possible to access parts of sequences, such as strings or lists. This makes it possible to access, modify, and delete items in a readable and concise fashion.

- Slicing works similar to indexing, but instead of accessing a single value, multiple values are accessed.

- Slicing uses indexing to access the range of elements. These indexes are also zero-based.

Slicing Syntax

You can slice iterables in Python in two possible ways:

`sequence[start:stop]`

- This way you access a range of elements beginning at index start and ending one before the stop index. In other words, it returns sequence[start], sequence[end – 1] and everything between.

`sequence[start:stop:step]`

- This means you get a range of elements from start to stop with a step size of the step. For example, with a step of 2, every second element in that range is returned.

For example, let's get the 4th, 5th, and 6th elements from a list of numbers:

```
nums = [1, 2, 3, 4, 5, 6, 7, 8]
part = nums[3:6]
print(part)
#Output:
[4, 5, 6]
```

23. What is a negative index in Python?

ANSWER:

- Negative indexes are a way to allow you to index into a list, tuple or other indexable container relative to the end of the container, rather than the start.The reason this is called negative indexing is that the number you are using is less than the 0th element in the list.

- Negative indexing in Python is when you use a negative number to index a list.

For example:

```
myList = ['A', 'B', 'C', 'D', 'E']
print(myList[-1])
#Output
E
```

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

ANSWER:

- a one-line version of the if-else statement. It provides a way to write conditional statements in a single line, replacing the multi-line if-else syntax.

Syntax of Python ternary operator:

```
<true_value> if <conditional_expression> else <false_value>
```

Python ternary operator works with three operands:

1.conditional_expression:
-This is a boolean condition that evaluates to either true or false.

Example:

```
>>> nice_weather = True
>>> print("Go out for a walk" if nice_weather else "watch a movie at home")
#Output
Go out for a walk
```

2.true_value: The value returned by the ternary operator if the conditional_expression evaluates to True.

Example:

```
>>> nice_weather = False
>>> print("Go out for a walk" if nice_weather else "watch a movie at home")
#Output
watch a movie at home
```

3. false_value: The value returned by the ternary operator if the conditional_expression evaluates to False.

Example:

```
>>> num = 12
>>> print("Even" if num % 2 == 0 else "Odd")
#Output
 Even
```

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

ANSWER:

- In Python, we can pass a variable number of arguments to a function using special symbols.

-Special Symbols Used for passing arguments:-

1.)*args (Non-Keyword Arguments)

2.)**kwargs (Keyword Arguments)

USES:

1.) *args The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

- The syntax is to use the symbol * to take in a variable number of arguments; by convention, it is often used with the word args.
- What *args allows you to do is take in more arguments than the number of formal arguments that you previously defined. With *args, any number of extra arguments can be tacked on to your current formal parameters (including zero extra arguments).
- For example : we want to make a multiply function that takes any number of arguments and able to multiply them all together. It can be done using *args.
- Using the *, the variable that we associate with the * becomes an iterable meaning you can do things like iterate over it, run some higher-order functions such as map and filter, etc.

2.) **kwargs The special syntax **kwargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is because the double star allows us to pass through keyword arguments (and any number of them).

- A keyword argument is where you provide a name to the variable as you pass it into the function.
- One can think of the kwargs as being a dictionary that maps each keyword to the value that we pass alongside it. That is why when we iterate over the kwargs there doesn't seem to be any order in which they were printed out.

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

ANSWER:

- The Range() or Xrange() functions in Python can be used in many parts of the program. They help to reduce the size of code and improve its readability.

DIFFERENCE BETWEEN Range() AND Xrange():

Attributes	Range()	Xrange()
Parameters	Numerical Values	Numerical Values
Return type	It gives a list of integers	It gives a generator object
Memory Utilization	It gives a list of elements and takes more memory	It takes less memory less range ()
Efficiency	Its execution time is slow	Its execution time is fast

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

ANSWER:

- Flask is a web framework. This means flask provides you with tools, libraries and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki or go as big as a web-based calendar application or a commercial website.
- Benefits of Flask Framework:
    - Integrated support for unit testing
    - Support for secure cookies (client-side sessions)
    - Easily deployable in production
    - Easy to create APIs
    - Supports Visual Debugging
    - Can be easily integrated with all the major databases

Netflix and Uber are a few such companies that use Flask as its backend.

Flask is easy to learn as well. There is extremely well-written documentation for the same. It is gaining popularity at a rapid pace.

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

ANSWER:

- 1. Clustered Index : Clustered index is created only when both the following conditions satisfy –
    - -The data or file, that you are moving into secondary memory should be in sequential or sorted order.
    - There should be a key value, meaning it can not have repeated values.

2. Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

- The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

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

ANSWER:

- A deadlock is a condition that occurs when two or more different database tasks are waiting for each other and none of the task is willing to give up the resources that other task needs.
- It is an unwanted situation that may result when two or more transactions are each waiting for locks held by the other to be released.
- In deadlock situation, no task ever gets finished and is in waiting state forever.
- Deadlocks are not good for the system

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

ANSWER:

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

2. Python string method and provide an example **29 points**

METHOD	DESCRIPTION	EXAMPLE
capitalize()	The capitalize() method returns a string where the first character is upper case, and the rest is lower case.	text = "my name is PAULA!" a = text.capitalize() print (a)
casefold()	The casefold() method returns a string where all the characters are lower case. This method is similar to the lower() method, but the casefold() method is stronger, more aggressive, meaning that it will convert more characters into lower case, and will find more matches when comparing two strings and both are converted using the casefold() method.	text = "Hello, How Are You TODAY?" a = text.casefold() print(a)
center()	The center() method will center align the string, using a specified character (space is default) as the fill character.	text = "orange" a = text.center(50) print(a)
count()	The count() method returns the number of elements with the specified value.	names = ["anna", "rachel", "linda"] a = names.count("rachel") print(a)
endswith()	The endswith() method returns True if the string ends with the specified value, otherwise False.	text = "Hello, my name is paula." a = text.endswith(".") print(a)
find()	The find() method finds the first occurrence of the specified value. The find() method returns -1 if the value is not found. The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the	text = "Hello, my name is paula." a = text.find("paula") print(a)

	value is not found. (See example below)	
format()	The format() method formats the specified value(s) and insert them inside the string's placeholder. The placeholder is defined using curly brackets: {}. Read more about the placeholders in the Placeholder section below. The format() method returns the formatted string.	text = "This is {price:.2f} zlotys only!" print(text.format(price = 49))
index()	The index() method returns the position at the first occurrence of the specified value.	names = ["anna", "rachel", "linda"] a = names.index("rachel") print(a)
isalnum()	The isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9). Example of characters that are not alphanumeric: (space)!#%&? etc.	text = "Paula0418" a = text.isalnum() print(a)
isdigit()	The isdigit() method returns True if all the characters are digits, otherwise False. Exponents, like ², are also considered to be a digit.	text = "1919234" a = text.isdigit() print(a)
isnumeric()	The isnumeric() method returns True if all the characters are numeric (0-9), otherwise False. Exponents, like ² and ¾ are also considered to be numeric values. "-1" and "1.5" are NOT considered numeric values, because all the characters in the string must be numeric, and the - and the . are not.	text = "1234567" a = text.isnumeric() print(a)
isspace()	The isspace() method returns True if all the characters in a string are whitespaces, otherwise False.	text = " " a = text.isspace() print(a)

istitle()	The istitle() method returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letters, otherwise False. Symbols and numbers are ignored.	text = "Hello, My Name Is Paula" a = text.istitle() print(a)
isupper()	The isupper() method returns True if all the characters are in upper case, otherwise False. Numbers, symbols and spaces are not checked, only alphabet characters.	text = "MY NAME IS PAULA!" a = text.isupper() print(a)
join()	The join() method takes all items in an iterable and joins them into one string. A string must be specified as the separator.	nameTuple = ("Paula", "Marie", "Jessica") a = "!".join(nameTuple) print(a)
lower()	The lower() method returns a string where all characters are lower case. Symbols and Numbers are ignored.	text = "Hello my name is PAULA" a = text.lower() print(a)
lstrip()	The lstrip() method removes any leading characters (space is the default leading character to remove)	text = " cherry " a = text.lstrip() print("of all fruits", a, "is my favorite")
replace()	The replace() method replaces a specified phrase with another specified phrase.	text = "I like bags" a = text.replace("bag", "shoes") print(a)
rsplit()	The rsplit() method splits a string into a list, starting from the right. If no "max" is specified, this method will return the same as the split() method.	names = "paula, bianca, aishee" a = names.rsplit(", ") print(a)
rstrip()	The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.	text = " cherry " a = text.rstrip() print("of all fruits", a, "is my favorite")

split()	The split() method splits a string into a list. You can specify the separator, default separator is any whitespace.	text = "my name is paula" a = text.split() print(a)
splitlines()	The splitlines() method splits a string into a list. The splitting is done at line breaks.	text = "I love you\nI hate you" a = text.splitlines() print(a)
startswith()	The startswith() method returns True if the string starts with the specified value, otherwise False.	text = "Hello, my name is paula." a = text.startswith("Hello") print(a)
strip()	The strip() method removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)	text = " cherry " a = text.strip() print("of all fruits", a, "is my favorite")
swapcase()	The swapcase() method returns a string where all the upper case letters are lower case and vice versa.	text = "Hello My Name Is PAULA" a = text.swapcase() print(a)
title()	The title() method returns a string where the first character in every word is upper case. Like a header, or a title. If the word contains a number or a symbol, the first letter after that will be converted to upper case.	text = "Welcome my name is paula" a = text.title() print(a)
upper()	The upper() method returns a string where all characters are in upper case. Symbols and Numbers are ignored.	text = "Hello my name is paula" a = text.upper() print(a)

3. Python list methods: **11 points**

describe each method and provide an example

METHOD	DESCRIPTION	EXAMPLE
append()	Adds an element at the end of the list	names = ["paula", "ann", "marie"] names.append("debbie") print(names)
clear()	Removes all the elements from the list	names = ["paula", "ann", "marie"] names.clear() print(names)
copy()	Returns a copy of the list	names = ["paula", "ann", "marie"] a = names.copy() print(a)
count()	Returns the number of elements with the specified value	
extend()	Add the elements of a list (or any iterable), to the end of the current list	names = ["paula", "ann", "marie"] a = names.count("marie") print(a)
index()	Returns the index of the first element with the specified value	names = ["paula", "ann", "marie"] a = names.index("marie") print(a)
insert()	Adds an element at the specified position	names = ["paula", "ann", "marie"] names.insert(1, "dalhia") print(names)
pop()	Removes the element at the specified position	names = ["paula", "ann", "marie"] names.pop(1) print(names)
remove()	Removes the first item with the specified value	names = ["paula", "ann", "marie"] names.remove("ann") print(names)
reverse()	Reverses the order of the list	names = ["paula", "ann", "marie"] names.reverse() print(names)
sort()	Sorts the list	names = ["paula", "ann", "marie"] names.sort() print(names)

4. Python tuple methods: **2 points**

describe each method and provide an example

METHOD	DESCRIPTION	EXAMPLE
count()	Returns the number of times a specified value occurs in a tuple	mytuple = (1, 5, 6, 9, 2, 5, 3, 9, 4, 5) a = mytuple.count(5) print(a)
index()	Searches the tuple for a specified value and returns the position of where it was found	mytuple = (1, 7, 9, 4, 8, 3, 4, 5, 2, 6) a = mytuple.index(8) print(a)

5. Python dictionary methods: **11 points**

describe each method and provide an example

METHOD	DESCRIPTION	EXAMPLE

clear()	Removes all the elements from the dictionary	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		car.clear()
		
		print(car)

copy()	Returns a copy of the dictionary	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.copy()
		
		print(x)

fromkeys()	Returns a dictionary with the specified keys and value	x = ('key1', 'key2', 'key3') y = 0 thisdict = dict.fromkeys(x, y) print(thisdict)

get()	Returns the value of the specified key	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.get("model")
		
		print(x)

items()	Returns a list containing a tuple for each key value pair	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.items()
		
		print(x)

keys()	Returns a list containing the dictionary's keys	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.keys()
		
		print(x)

pop()	Removes the element with the specified key	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		car.pop("model")
		
		print(car)

popitem()	Removes the last inserted key-value pair	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		car.popitem()
		
		print(car)

setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.setdefault("model", "Bronco")
		
		print(x)

update()	Updates the dictionary with the specified key-value pairs	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		car.update({"color": "White"})
		
		print(car)

values()	Returns a list of all the values in the dictionary	car = {
		"brand": "Ford",
		"model": "Mustang",
		"year": 1964
		}
		
		x = car.values()
		
		print(x)

6. Python set methods: **12 points**

describe each method and provide an example

METHOD	DESCRIPTION	EXAMPLE
add()	Adds an element to the set	thisset = {"apple", "banana", "cherry"} thisset.add("orange") print(thisset)
clear()	Removes all the elements from the set	thisset = {"apple", "banana", "cherry"} thisset.clear() print(thisset)
copy()	Returns a copy of the set	fruits = {"apple", "banana", "cherry"} x = fruits.copy() print(x)

difference()	Returns a set containing the difference between two or more sets	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.difference(y) print(z)
intersection()	Returns a set, that is the intersection of two or more sets	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.intersection(y) print(z)
issubset()	Returns a set, that is the intersection of two or more sets	x = {"a", "b", "c"} y = {"f", "e", "d", "c", "b", "a"} z = x.issubset(y) print(z)
issuperset()	Returns whether this set contains another set or not	x = {"f", "e", "d", "c", "b", "a"} y = {"a", "b", "c"} z = x.issuperset(y) print(z)
pop()	Removes an element from the set	fruits = {"apple", "banana", "cherry"} fruits.pop() print(fruits)
remove()	Removes the specified element	fruits = {"apple", "banana", "cherry"} fruits.remove("banana") print(fruits)
symmetric_difference()	Returns a set with the symmetric differences of two sets	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.symmetric_difference(y) print(z)

union()	Return a set containing the union of sets	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} z = x.union(y) print(z)
update()	Update the set with another set, or any other iterable	x = {"apple", "banana", "cherry"} y = {"google", "microsoft", "apple"} x.update(y) print(x)

7. Python file methods: **5 points**

describe each method and provide an example

METHOD	DESCRIPTION	EXAMPLE
read()	Returns the file content	f = open("demofile.txt", "r") print(f.read())
readline()	Returns one line from the file	f = open("demofile.txt", "r") print(f.readline())
readlines()	Returns a list of lines from the file	f = open("demofile.txt", "r") print(f.readlines())

write()	Writes the specified string to the file	f = open("demofile2.txt", "a")
		f.write("See you soon!")
		f.close()
		
		#open and read the file after the appending:
		f = open("demofile2.txt", "r")
		print(f.read())

writelines()	Writes a list of strings to the file	f = open("demofile3.txt", "a")
		f.writelines(["See you soon!", "Over and out."])
		f.close()
		
		#open and read the file after the appending:
		f = open("demofile3.txt", "r")
		print(f.read())