Python theory questions

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

Python is a dynamic programming language that is used in making software, creating automation scripts, and used in manipulating and analysis of data. The main features of python include

- It is a dynamic language, therefore the data types of variables are decided at runtime.
- It is object-oriented
- It is easy to learn, making it a favorite for beginners
- It is an interpreted language and does not require compilation.

2. Discuss the difference between Python 2 and Python 3

- Python 3 syntax is easier to understand while Python 2 syntax is a bit complicated
- In python 2, printing to the console was done with the print statement without parenthesis while in python 3, printing to the console is done with the print function which includes parentheses.
- In python 2, integer division is done using the single slash operator like so “/” while in python 3, integer division is done using a double slash operator like so “//”.
- In python 2, strings are ASCII bytes by default while strings are Unicode by default in python 3.
- In python 2 exception handling, the exception message is not included inside parentheses but in python 3, the exception message must be inside a parentheses

3. What is PEP 8?

PEP 8 is a style guide document that outlines best practices for writing clean and readable python code.

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

A computer program is a set of instructions written in a programming language and these instructions tell the computer how to carry out a task.

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

A process refers to a computer program that is currently being executed or run.

6. In computing/computer science what is cache?

A cache is a storage area that holds temporary files that are often used so that retrieval of such files can be done faster and efficiently without eating up memory.

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

A thread is a part of a process. It refers to a path in the execution of a program(process). Every process has a thread. A process can have more than one thread and this refers to multi-threading. To successfully execute a program, a process can be divided into multiple threads, with each thread containing instructions for the execution of a program.

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

Concurrency is a process that occurs when a computer executes multiple tasks in an overlapping, non-sequential manner. The computer can start a task and mid-way switch context and start executing another task. Parallelism is a process that occurs when a computer independently executes the tasks on a program at the same time.

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

GIL stands for Python Global Interpreter Lock and it refers to a process lock that causes python to execute only one thread at a time. GIL prevents multi-threading in python in order to preserve the accuracy of the reference counter of objects.

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

DRY: Do not repeat yourself. It refers to avoiding unnecessary code repetition.
KISS: Keep it simple, stupid. It refers to keeping things uncomplicated, simple and efficient.
BDUF: Big design up front. It means that before a software is created, it should be designed beforehand and the design should be thorough and efficient.

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

A garbage collector is what frees up memory when an object is no longer referenced. When an object is initialized, python keeps a reference count of the object. When the object is no longer in scope and the reference count is zero, python removes it from memory using the garbage collector.

12. How is memory managed in Python?

Memory is managed in python by garbage collection and reference counting. References to objects are counted and when there are no more references to an object, the object is deallocated from memory by the garbage collector and memory is freed up.

13. What is a Python module?
This is a file containing python code. Files containing programs written in python are modules.

14. What is docstring in Python?

A concise string of information that documents a module, class, or function.

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

Pickling is a process in which python objects are converted to byte streams and unpickling is the reverse of pickling in which the byte streams are reconstructed to produce the original python object.
Example pickling

```
import pickle

student = {
    "firstName": "Mary",
    "lastName": "John",
    "email": "maryjohn@gmail.com",
    "phoneNumber": "555-555-5555"
}


with open("data.pickle","wb") as my_file:
    pickle.dump(student, my_file, pickle.HIGHEST_PROTOCOL)
```

Example unpickling

```
import pickle

with open("data.pickle","rb") as my_file:
   student = pickle.load(my_file)
   print(student)
```

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

Assertions and debugger breakpoints.

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

Arguments are passed by reference. Example

```
fruits = ["cherry", "apple", "orange"]
def buy_fruits():
   fruit = "pineapple"
   fruits.append(fruit)
   print(fruits)
   return fruits
```

18. What are Dictionary and List comprehensions in Python? Provide examples.
Dictionary and list comprehensions are a way of generating a new dictionary or new list respectively, in a more concise and elegant way without using the traditional for loop method.
Example list comprehension

```
numbers = [num for num in range(10)]
```

Example of dictionary comprehension

```
mydict = {i : i+1 for i in range(10)}
print(mydict)
```

19. What is namespace in Python?
A namespace is a collection of names, in which each object is mapped to a defined name.

20. What is pass in Python?
It is a null statement that tells the Python interpreter to do nothing when encountered.

21. What is unit test in Python?
A test that analyzes the units of a module to ensure that it is working correctly.

22. In Python what is slicing?
Slicing is a way to extract a part of a string, list, or tuple.

23. What is a negative index in Python?
This is a feature of python that allows us to access the elements of a list or tuple from the end, rather than from the start.

24. How can the ternary operators be used in python? Give an example.
A ternary operator in python uses this syntax

```
(true_value) if (conditional_expression) else (false_value)
```

Example

```
sick = True
a = "Go to the hospital"
b = "Stay at home"
print(a if sick else b)
```

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

*args are used as function arguments when we need to pass any number of non-keyword arguments to a function. It is used when you need to pass a variable number of arguments to a function.

**kwargs are used as function arguments when we need to pass any number of keyword arguments to a function. It is used when you need to pass a variable number of keyword arguments to a function.

26. How are range and xrange different from one another?
range function is available in python 3 and python 2 while xrange function is only available in python 2.

27. What is Flask and what can we use it for?
Flask is a python framework for creating web applications.

28. What are clustered and non-clustered index in a relational database?
A clustered index is an index that defines the order in which data is added to a table and it speeds up the retrieval of records upon querying. Non-clustered index is an index that is separate from the table. It is defined based on certain columns that tend to be queried a lot and it hastens retrieval of records when such columns are queried.

29. What is a ‘deadlock’ in a relational database?
It is a situation in which two or more transactions are holding on to locks on a table and waiting endlessly for the other to give up the locks.

30. What is a ‘livelock’ in a relational database?
It is a situation in which a request for an exclusive lock is denied repeatedly, because of interference from many overlapping shared locks.

2. Python string methods

capitalize() : a method that converts the first character in a string to uppercase and converts the rest of the characters to lowercase. It returns a new copy of the string.

Example:

```
country = "ameRICA"
print(country.capitalize())
```

casefold(): converts all characters of a string to lowercase and returns a new copy of the string.
Example:

```
item = "LuGGage"
print(item.casefold())
```

center(): returns a new centered string.It takes a length which defines the length of the returned string.
Example:

```
greeting = "Welcome"
print(greeting.center(40))
```

count(): returns the number of times a value occurs in a given string.
Example:

```
like = "I like to eat ice-cream but I do not like to eat broccoli"
print(like.count("eat"))
```

endswith(): returns True if a string ends with a specified value, otherwise it returns false.
Example:

```
like = "I like to eat ice-cream but I do not like to eat broccoli"
print(like.endswith("broccoli"))
```

find(): returns the index of the first occurrence of a specified value in a string. It returns -1 if the value does not exist in the string.
Example:

```
email = "mary@gmail.com"
print(email.find("@"))
```

format(): It is used to format a string using placeholders for a nicer, readable output.
Example:

```
candidate = "Dear {name}, you have been offered a role in the department of
{department} ".format(name="Jon Snow", department="Aragon")
print(candidate)
```

index(): Given a value, it returns the index of that value in a string and an error if the value is not found in the string.
Example:

```
question = "What is your name"
print(question.index("What"))
```

isalnum(): returns true is a string is made up of only alphanumeric characters, otherwise, it returns false.
Example:

```
str = "address2"
print(str.isalnum())
```

isalpha(): returns true is a string is made up of only alphabets.
Example:

```
str = "address"
print(str.isalpha())
```

isdigit(): returns true is a string is made up of only digits.
Example:

```
str = "12029373"
print(str.isdigit())
```

islower(): returns true is a string is made up of only lowercase characters.
Example:

```
item = "luggage"
print(item.islower())
```

isnumeric(): returns true is a string is made up of only numeric characters.
Example:

```
str = "1202938"
print(str.isnumeric())
```

isspace(): returns true is a string is made up of only numeric whitespaces.
Example:

```
empty = "    "
print(empty.isspace())
```

istitle(): returns true if a string is formatted as a title, with each word in the string being in uppercase while the rest being in lowercase.
Example:

```
str = "His royal highness"
print(str.istitle())
```

isupper(): returns true is a string is made up of only uppercase characters.
Example:

```
str = "BROADWAY"
print(str.isupper())
```

join(): converts the elements of a list or tuple into a string.
Example:

```
my_list = ["cherry", "apples", "nuts"]
print(", ".join(my_list))
```

lower(): converts all characters of a string to lowercase.
Example:

```
str = "WE ARE GOING HOME"
print(str.lower())
```

lstrip(): removes all whitespaces at the beginning of a string.
Example:

```
str = "     pancakes are nice for breakfast"
print(str.lstrip())
```

replace(): Used to replace a part of a string with a specified value.
Example:

```
str = "the sky is red"
print(str.replace("red", "blue"))
```

rsplit(): Converts a string to a list by splitting it using a specified value.
Example:

```
str = "bags, shoes, glasses"
print(str.rsplit(", "))
```

rstrip(): removes all whitespaces at the end of a string.
Example:

```
str = "where are you going?                "
print(str.rstrip())
```

split(): Converts a string to a list by splitting it using a specified value.
Example:

```
str = "bags, shoes, glasses"
print(str.split())
```

splitlines(): Splits a string at new line characters and returns a list.
Example:

```
str = "My name is Mary.\nI am a student"
print(str.splitlines())
```

startswith(): Returns true if a string begins with a specified value, otherwise, returns false.

```
str = "My name is Mary.\nI am a student"
print(str.startswith("My"))
```

strip(): Removes whitespaces at the beginning and end of a string.
Example:

```
str = "     What a wonderful world     "
print(str.strip())
```

swapcase(): Converts lowercase characters to uppercase and converts uppercase characters to lowercase.
Example:

```
str = "kEEp iT SimPLe"
print(str.swapcase())
```

title(): Converts the first character of each word in a string to upper case.
Example:

```
str = "hello world"
print(str.title())
```

upper(): converts all characters of a string to uppercase.
Example:

```
str = "hello world"
print(str.upper())
```

3. Python list methods

append(): Adds an element to the end of a list

```
items = ["shoes", "bags"]
items.append("glasses")
print(items)
```

clear(): Removes all the elements in a list

```
items = ['shoes', 'bags', 'glasses']
items.clear()
print(items)
```

copy(): Returns a copy of the list

```
items = ['shoes', 'bags', 'glasses']
items2 = items.copy()
print(items2)
```

count(): Returns the number of times a specified element occurs in a list

```
items = ['shoes', 'bags', 'glasses', 'bags']
print(items.count('bags'))
```

extend(): Takes the items in a list and adds them to the end of a specified list

```
items = ['shoes', 'bags', 'glasses']
items2 = ['rings', 'shirts']
items.extend(items2)
print(items)
```

index(): Given a specified value, it returns the index of the first occurrence of the value in the list.

```
items = ['shoes', 'bags', 'glasses']
print(items.index('bags'))
```

insert(): Adds a specified element to a list at the specified index.

```
items = ['shoes', 'bags', 'glasses']
items.insert(0,'ring')
print(items)
```

pop(): Removes the element located at the specified index.

```
items = ['shoes', 'bags', 'glasses']
items.pop(0)
print(items)
```

remove(): Removes the first occurrence of a specified value from the list.

```
items = ['shoes', 'bags', 'glasses']
items.pop(0)
print(items)
```

reverse(): Reverse the order of a string in place.

```
numbers = [1, 2, 3, 4, 5, 6]
numbers.reverse()
print(numbers)
```

sort(): Sorts a list based on a given condition, otherwise it sorts it in ascending order by default.

```
numbers = [6, 5, 4, 3, 2, 1]
numbers.sort()
print(numbers)
```

4. Python tuple methods

count(): Returns the number of times a given value occurs in a tuple.

```
numbers = (6, 5, 6, 4, 3, 2, 1)
print(numbers.count(6))
```

index(): Returns the index of the first occurrence of specified value.

```
numbers = (6, 5, 6, 4, 3, 2, 1)
print(numbers.index(6))
```

5. Python Dictionary methods

clear(): Removes all the elements in a dictionary

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
mydict.clear()
print(mydict)
```

copy(): Returns a copy of the specified dictionary.

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
mydict_b = mydict.copy()
print(mydict_b)
```

fromkeys(): Creates a dictionary from a given sequence.

```
sequence = {"item1", "item2", "item3"}
value = "out of stock"
mydict = dict.fromkeys(sequence, value)
print(mydict)
```

get(): Returns the value of a specified dictionary key.

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
print(mydict.get("role"))
```

items(): Returns a list of tuples with each tuple containing key, value pair.

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
print(mydict.items())
```

keys(): Returns a list of a given dictionary's keys

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
print(mydict.keys())
```

pop(): Given a key, it removes the key-value pair whose key matches the specified key.

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
mydict.pop('name')
print(mydict)
```

popitem(): Removes the last key-value pair in a dictionary.

```
mydict = {"name": "Larry Page", "age": 30, "role": "Writer"}
mydict.popitem()
print(mydict)
```