1. Python is an interpreted and interactive programming language that is typically used to build websites and software, automate tasks and conduct data analysis.
2. Python 2's syntax is harder to understand whereas python 3 is easier to understand and this is evident in the rules of ordering being simplified in python 3 whereas in python 2, this is much more difficult.
3. PEP 8 stands for Python Enhancement proposals and is a document that is used to guide users on the best practices in writing python code.
4. In computer science, a program is a set of ordered operations for a computer to execute.
5. In computer science, a process is the instance in which a program is executed and contains the program code as well as activity
6. In computer science, a cache is a small amount of memory that stores temporary files by use of hardware and software elements.
7. In computer science, a thread is a path of execution within a process. Multithreading is the ability of a program or an operating system to execute multiple threads to be created within a process.
8. In computer science, concurrency is the ability of different units of a program, problem or algorithm ti be executed out-of-order or in partial order, without compromising the final outcome. Parallelism is a type of computation in which many calculations or processes are carried out simultaneously. The aim is usually to make the program faster.
9. GIL in python is a lock that allows only one thread to hold the control of the python interpreter
10. DRY stands for 'don't repeat yourself' and is intended to be a Principe that reduces the repetition of software patterns or information. KISS stands for 'keep it simple stupid' and is also a Principe intended to state that most systems wok best when they are kept simple rather than complex. BDUF means 'big design up front' which is an approach in which the program's design is to be completed and perfected before that program's implementation is started.
11. A Garbage collector in python has a threshold of a number of objects per generation of which there is three. An object does into an older generation when it survives a garbage collection process.
12. Memory is managed/allocated through a private heap containing all python objects and data structures. This is then ,managed by the python memory manager
13. A python module is a code library or a file that includes functions that the user might include in their application and will be evident through the ".py" extension.
14. A docstring in python is a string yard to document a python module, class, function or method.
15. Pickling is the process in which a python object hierarchy is converted int a byte stream and unpicking would be the opposite operation. For example in pre-processing where routine tasks maybe performed.
16. Pychecker and Pylint are the static analysis tools that help to find bugs in python.
17. Arguments are passed through by reference. When the user changes what a parameter refers to within a function, the change also reflects back in the calling function. E.g
    def my_function(**fname**):
18. print(fname + " steward")

my_function(**"Jacqueil"**)
my_function(**"Tobias"**)
my_function(**"Steph"**
19. Dictionary comprehension stores data as key-value pairs. List comprehension create new lists from other iterables such as tupelos, strings in a more concise format.
20. Namespace in python is a way of providing unique name for each object in python.
21. A pass in python is used as a placeholder for instance when the user does not know what code to write.
22. Slicing allows the user to return a range of characters
23. A negative index in python indexes from the end and the last value of the sequence will have a '-1' and so on as the user goes along.
24. The ternary operators are used to with conditional statements e.g is_bad = **True**
state = "nice" **if** is_bad **else** "not nice"
25. Args is used to pass a variable number of non-key word arguments to a function.
Kwargs is used to passs a variable length of key-word arguments to a function. The can be used in instances where the user is unsure of how many arguments to pass through the function.
26. Range returns an immutable sequence of numbers. XRange () does the same as range but is used in python 2

27. Flask us a micro web framework written in python and is meant to be easier in terms of productivity as I provides the user with the tools, libraries and technologies that allow you to build a web application.
28. A clustered index is physically reordered to match the index whereas a non-clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk.
29.A deadlock relational database occulted when two or more transactions are waiting for another to give up locks.
30. A livelock relational database occurs when a request for an exclusive lock is denied continuously because a series of overlapping shared locks keep on interfering.

2.

1. Capitalize() converts the first character of a string to an upper case letter and all other alphabets to lowercase.

E.g.

```
Txt = "bounjour, and let the test begin."
x = txt.capitalize()
print (x)
```

2. Casefold() returns a string so that all the characters are lowercase

E.g.

```
ThisTheoryAssignmentIsEasy.casefold()
```

3. Center () center aligns the string through using a specified character as the fill character e.g.

```
txt = “Pineapple"
x = txt.center(20)
print(x)
```

4. Count() returns the number of elements with the specified value e.g.

```
numbers = [2, 3, 5, 2, 11, 2, 7]
count = numbers.count(2)
print('Count of 2:', count)
```

5. endwith() returns True if the strong ends with the specified value and false if it does not.

```
txt = "wow, my goodness.”
x = txt.endswith("my goodness.”)
print(x)
```

6. Find() finds the first occurrence of the specified value
e.g.

```
message = 'Python is a fun programming language’

print(message.find(‘fun’))
```

7. Fomat() method formats the specified value(s) and insert them inside the string's placeholder.The placeholder is defined using curly brackets: {}

E.g.

```
>>> print('{0} {1} cost ${2}'.format(6, 'bananas', 1.74))
6 bananas cost $1.74
```

8. Index() returns the index of the specified element in the list

e.g. animals = ['cat', 'dog', 'rabbit', ‘horse']
index = animals.index(‘dog')
print(index)

The output will be 1.

9. isalnum() returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9).

e.g.

txt = “Company 14”
x = txt.isalnum()
print(x)

This will print False.

10. isalpha() returns True if all the characters are alphabet letters (a-z)

e.g.

txt = “Companyp”
x = txt.isalpha()
print(x)

This will print true.

11. isdigit() returns True if all the characters are digits, otherwise False.

e.g.

txt = “220601”
x = txt.isdigit()
print(x)

This will print true.

12. islower() returns True if all the characters are in lower case, otherwise False.

e.g.

txt = “hello instructors!”
x = txt.islower()
print(x)

This will print

13. isnumeric() returns True if all the characters are numeric (0-9), otherwise False.
e.g.

a = “\u0030"
print(a.isnumeric())

14. isspace() returns True if all the characters in a string are whitespaces, otherwise False.

e.g.

```
txt = “ s   “
x = txt.isspace()
print(x)
```

This will return false.

15. istitle() 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.

e.g.

```
b = “Hello”
print(b.istitle())
```

This will print true

16. isupper() returns True if all the characters are in upper case, otherwise False.

e.g.

```
c = "MY NAME IS TRICIA”
print(c.isupper())
```

This wil return true

17. join() method takes all items in an iterable and joins them into one string.

e.g.

```
text = ['Python', 'is', 'a', 'fun', 'programming', ‘language']
print(' ‘.join(text))
```

The output will be: Python is a fun programming language

18. lower() returns a string where all characters are lower case.

```
txt = "Hello my AMAzInG inStrUctoRs”
x = txt.lower()
print(x)
```

19. lstrip() removes any leading characters.

e.g.

```
txt = "     banana     “
x = txt.lstrip()
print("of all fruits", x, "is my favorite”)
```

Output = of all fruits banana   is my favorite

20. replace() replaces a specified phrase with another specified phrase.

e.g.

```
txt = "I like bananas”
x = txt.replace("bananas", “cherries")
print(x)
```

Ouput= I like cherries

21. rpslit() splits a string into a list, starting from the right.

E,g.

txt = "apple, banana, cherry”
x = txt.rsplit(", “)
print(x)
output= ['apple', 'banana', ‘cherry']

22. rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.

e.g.

txt = “chicken,,,,,ssqqqww....."
x = txt.rstrip(“,.qsw")
print(x)

Ouput= chicken

23. split() method splits a string into a list.

e.g.

txt = "hello, my name is Tricia, I am 21 years old”
x = txt.split(", “)
print(x)

Output= ['hello', 'my name is Tricia', 'I am 21 years old’]

24. splitlines() splits a string into a list. The splitting is done at line breaks.

e.g.

txt = "Thank you for the assignment\nWelcome to the reward"

x = txt.splitlines(True)

print(x)

Ouput= ['Thank you for the assignment\n', 'Welcome to the reward']

25. startsiwth() returns True if the string starts with the specified value, otherwise False.

e.g.
txt = "Hello, welcome to my time.”
x = txt.startswith("wel", 7, 20)
print(x)

Output = True

26. strip() removes any leading (spaces at the beginning) and trailing (spaces at the end) characters (space is the default leading character to remove)

e.g.
txt = “   lemon   “
x = txt.strip()
print(“of all fruits", x, "is my favorite”)

Output= of all fruits lemon is my favorite

27. swapcase() returns a string where all the upper case letters are lower case and vice versa.
e.g.

```
string = "ThIs ShOuLd Be MiXeD cAsEd.”
print(string.swapcase())
```

Output- tHiS sHoUlD bE mIxEd CaSeD.

28. title() returns a string where the first character in every word is upper case. Like a header, or a title.

e.g.

```
txt = "Welcome to my 3rd world”
x = txt.title()
print(x)
```

Ouput-= Welcome To My 2Rd World

29. upper() returns a string where all characters are in upper case.

e.g.

```
txt = "Hello beautiful ”
x = txt.upper()
print(x)
```

output= HELLO BEAUTIFUL

30. append() appends an element to the end of the list.

```
a = ["apple", "banana", “cherry"]
b = ["Ford", "BMW", “Volvo"]
a.append(b)
```

Output= ['apple', 'banana', 'cherry', ["Ford", "BMW", “Volvo"]]

31. clear() removes all items form the list.

E.g.

```
prime_numbers = [2, 3, 5, 7, 9, 11]
prime_numbers.clear()
print(‘List after clear():', prime_numbers)
```

Output: List after clear(): []

32. copy() returns a copy of the specified list.

E.g:

```
prime_numbers = [2, 3, 5]
numbers = prime_numbers.copy()
print('Copied List:', numbers)
```

Output: Copied List: [2, 3, 5]

33. count()

returns the number of elements with the specified value.

e.g.
fruits = ['apple', 'banana', ‘cherry']
x = fruits.count(“cherry")

Output=1

34. extend() adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

e.g.

prime_numbers = [2, 3, 5]
numbers = [1, 4]
numbers.extend(prime_numbers)
print('List after extend():', numbers)
Output: List after extend(): [1, 4, 2, 3, 5]

35. insert() inserts an element to the list at the specified index.

vowel = ['a', 'e', 'i', ‘u’]
vowel.insert(3, ‘o')
print('List:', vowel)

Output: List: ['a', 'e', 'i', 'o', ‘u']

36. pop() removes the item at the given index from the list and returns the removed item.

e.g.

prime_numbers = [2, 3, 5, 7]
removed_element = prime_numbers.pop(2)

print('Removed Element:', removed_element)
print(‘Updated List:', prime_numbers)

Output= Removed Element: 5
Updated List: [2, 3, 7]

37. remove() removes the first matching element (which is passed as an argument) from the list.

prime_numbers = [2, 3, 5, 7, 9, 11]
prime_numbers.remove(9)
print('Updated List: ', prime_numbers)

Output= Updated List: [2, 3, 5, 7, 11]

38. reverse() reverses the elements of the list

e.g.

prime_numbers = [2, 3, 5, 7]
prime_numbers.reverse()
print('Reversed List:', prime_numbers)
Output= Reversed List: [7, 5, 3, 2]

39. count() method returns the number of times a specified value appears in the tuple.
e.g. thistuple = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)

```
x = thistuple.count(5)
print(x)
```

Output= 1

40. Index () method searches for the given element in a tuple and returns its position. It returns first occurrence of the element in the tuple.

e.g.

```
val = ('a','b','r','a','k','a','d','h','a')
print(val)
index = val.index('k')
print("Index of k is: ",index)
```

Output= ('a', 'b', 'r', 'a', 'k', 'a', 'd', 'h', ‘a')
Index of k is: 4

41.

clear() in a dictionary removes all items from the dictionary

E.g.

```
Input : d = {1: "geeks", 2: “for"}
            d.clear()
Output :    d = {}
```

42. copy() returns a copy (shallow copy) of the dictionary.

```
e.g. original = {1:'one', 2:’two'}
new = original.copy()
print('Orignal: ', original)
print('New: ', new)
```

Ouput= Orignal: {1: 'one', 2: ‘two'}
New: {1: 'one', 2: 'two'}

43. fromkeys() returns a dictionary with the specified keys and the specified value.
e.g.

```
x = ('key1', 'key2', ‘key3')
y = 0
thisdict = dict.fromkeys(x, y)
print(thisdict)
```

Ouput= ['key1': 0, 'key2': 0, 'key3': 0]

44. get() returns the value of the item with the specified key.

```
car = {
“brand”: “Ford"
"model": “Mustang",
"year": 1964
}
x = car.get(“model")
print(x)
```

Output= Mustang

45. items() returns a view object

E.g car = {
"brand": “Ford",
"model": “Mustang",
"year": 1964
}
x = car.items()
car["year"] = 2018
print(x)

output= dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 2018)])

46. keys() returns a view object. The view object contains the keys of the dictionary, as a list.

car = {
"brand": “Ford",
"model": “Mustang",
"year": 1964
}
x = car.keys()
print(x)

dict_keys(['brand', 'model', ‘year'])

47. pop() removes an element from the dictionary. It removes the element which is associated to the specified key.

e.g.

inventory = {'shirts': 25, 'paints': 220, 'shock': 525, 'tshirts': 217}
element = inventory.pop('shirts')
print(element)

Ouput=25

48. popitem() removes the item that was last inserted into the dictionary. In versions before 3.7, the popitem() method removes a random item.

car = {
"brand": “Ford",
"model": “Mustang",
"year": 1964
}
x = car.popitem()
print(x)

Ouput= ('year', 1964)

49. setdefault() returns the value of the item with the specified key. If the key does not exist, insert the key, with the specified value, see example below

car = {
 "brand": “Ford",
"model": “Mustang",
"year": 1964
}
x = car.setdefault("model", “Bronco")
print(x)
Output= Mustang

50. update()
updates the dictionary with the elements from another dictionary object or from an iterable of key/ value pairs.

E,g,

```
marks = {'Physics':67, ‘Maths':87}
internal_marks = {‘Practical':48}
marks.update(internal_marks)
print(marks)
```

Ouput= {'Physics': 67, 'Maths': 87, 'Practical': 48}

51. values() returns a view object that displays a list of all the values in the dictionary.

E.g.

```
marks = {'Physics':67, ‘Maths':87}
print(marks.values())
```
Output= dict_values([67, 87])

52. add() The set add() method adds a given element to a set if the element is not present in the set.

```
thisset = {"apple", “raspberries", “cherry"}

thisset.add(“orange")

print(thisset)
```

Output= {'cherry', 'banana', 'raspberries', 'apple'}

53. The clear() method removes all elements from the set.
e.g.

```
vowels = {'a', 'e', 'i', 'o', ‘u'}
print('Vowels (before clear):', vowels)
# clearing vowels
vowels.clear()
print('Vowels (after clear):', vowels)
```

Ouput= Vowels (before clear): {'e', 'a', 'o', 'u', ‘i'}
Vowels (after clear): set()

54. The copy() method returns a shallow copy of the set in python

55. difference() returns the set difference of two sets.

```
E.g. A = {'a', 'b', 'c', ‘d'}
B = {'c', 'f', ‘g'}
# Equivalent to A-B
print(A.difference(B))
# Equivalent to B-A
print(B.difference(A))
```
Output = {'b', 'a', ‘d'}
{'g', 'f'}

56. intersection() returns a set that contains the similarity between two or more sets.

e.g. x = {"a", "b", “c"}
y = {"c", "d", “e"}
z = {"f", "g", “c"}
result = x.intersection(y, z)
print(result)

Ouput= {‘c'}

57. issubset() returns True if all items in the set exists in the specified set, otherwise it retuns False.

e.g. x = {"a", "b", “c"}
y = {"f", "e", "d", "c", “b"}
z = x.issubset(y)
print(z)

Ouput= False

58. issuperset() returns True if a set has every elements of another set (passed as an argument). If not, it returns False.

e.g.
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}
C = {1, 2, 3}
# Returns True
print(A.issuperset(B))
# Returns False
print(B.issuperset(A))
# Returns True
print(C.issuperset(B))

59. pop() method removes a random item from the set.This method returns the removed item.

fruits = {"apple", "banana", “cherry"}
x = fruits.pop()
print(x)

60. remove() removes the specified element from the set.

languages = {'Python', 'Java', ‘Spanish’}
languages.remove(‘Spanish')
print(languages)

Ouput= {'Python', ‘Java'}

61.symmetric_difference() returns the symmetric difference of two sets.

e.g. A = {'a', 'b', 'c', ‘d'}
B = {'c', 'd', 'e' }
C = {}
print(A.symmetric_difference(B))
print(B.symmetric_difference(A))
print(A.symmetric_difference(C))
print(B.symmetric_difference(C))

Output =
{'b', 'a', ‘e'}
{'b', 'e', ‘a'}
{'b', 'd', 'c', ‘a'}

{'d', 'e', ‘c'}

62. union() returns a set that contains all items from the original set, and all items from the specified set(s).

e.g.
x = {"a", "b", “c"}
y = {"f", "d", “a"}
z = {"c", "d", “e"}
result = x.union(y, z)
print(result)

Output= {'d', 'e', 'c', 'a', 'f', ‘b'}

63. update() updates the set, adding items from other iterables.

e.g.
A = {'a', ‘b'}
B = {1, 2, 3}
result = A.update(B)
print('A =', A)
print('result =', result)]

Output=
A = {'a', 1, 2, 3, ‘b'}
result = None

64. read() returns the whole text, but you can also specify how many characters you want to return:

E,g.

f = open(“examplefile.txt", “r")
print(f.read(**6**))

65. readline() returns one line from the file.

e.g. f = open(“Tricia.txt", “r")
print(f.readline()
print(f.readline())

66. readlines() returns a list containing each line in the file as a list item.

f = open("Tricia.txt", “r")
print(f.readlines(21))

67. write() writes a specified text to the file

e.g.
f = open("demofile2.txt", “a")
f.write("\nSee you soon!”)
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", “r")
print(f.read())

68. Writelines() method writes the items of a list to the file.

e.g.

```
f = open("demofile3.txt", “a")
f.writelines(["\nSee you soon!", "\nOver and out.”])
f.close()

#open and read the file after the appending:
f = open("demofile3.txt", “r")
print(f.read())
```