**Theory questions**

1. Python is a computer programming language. It’s main features include :

* it’s price (free),
* it’s accessibility (open source),
* it’s ease (resembles syntax of spoken/written English language better than other programming languages),
* it’s portability (works the same on Linux/Mac/Windows etc.),
* it’s interpretation (done line by line, making it easy to detect bugs).

2. There are many differences between python 2 and 3, including:

* syntax, where python 3 is must easier to understand
* the applicable libraries, where newer libraries are only compatible with python 3
* division of integers, where python 3 returns a float (e.g., 7/2=3.5) and python 2 returns an integer (7/2=4)

3. PEP 8 is a document written in 2001 about the best practices for writing code in python. PEP stands for Python Enhancement Proposal, and PEP-8 is the newest version. It is aimed at people who are beginners/intermediates in coding, and aims to make programmers write code that is readable and consistent.

4. A program is a specific set of ordered operations for a computer to perform. The program contains a line by line sequence of instructions that the computer follows in order to provide an outcome.

5. A process is a program in execution.

6. Cache is a small amount of memory which is a part of the computer processing unit - closer than RAM . It is used to temporarily hold instructions and data that the CPU is likely to reuse.

7. Threading refers to the smallest process (i.e., executing a program) possible. To multithread means to execute multiple threads at once, independently i.e., running multiple parts of a program at the same time. Multithreading is essentially multitasking and maximises utility in the computer.

8. Concurrently means to execute multiple tasks at once (i.e., it is not the case that you cannot move onto the next task until you complete the task at hand). To execute 2 tasks concurrently does not necessarily mean simultaneously (the computer may do some of task 1, then some of task 2, then the rest of task 1, then the rest of task 2). Parallelism is a specific kind of concurrency where tasks are always executed simultaneously. To sum concurrency is about dealing with many things at once (which may or may not mean executing them simultaneously), whereas parallelism is about doing many things at once.

9. GIL stands for Global Interpreter Lock. It means that the interpreter of the code (i.e., Python) can only execute 1 thread at any point in time, and cannot deal with or execute multiple threads (i.e., it cannot show concurrency or parallelism)

10. DRY = don’t repeat yourself. This means that any important bit of information should have only 1 corresponding piece of code. This is so that if the information needs to be changed, all other instances of the information in the code will change automatically.

KISS = keep it simple stupid. It means that simplicity should be a goal in code and complexity which is not necessary should be avoided.

BDUF = big design up front. This means the program design should be fully complete before the implementation stage in development. It is linked to the waterfall method, where workers do not move on to the next stage until the current stage is complete.

11. Garbage collection is the process of freeing and reclaiming memory. The garbage collector runs during a process and if it detects an object (a variable or function) that is not referenced in the code at all, then the object will be deleted and won’t unnecessarily take up memory any longer.

12. The python memory manager is a complex feature of python that manages the heap of all the memory used in python

13. A module is the name of any file in python that contains information of some kind. E.g., when making a .txt. or .py file, each of these files can be referred to as a module

14. A docstring is essentially a comment in python, which serves the function of explaining the code associated with it

15. Pickling involves a Python object hierarchy (e.g., the language used in python) being converted into a byte stream (I.e., the binary numbers of 1 and 0 that are recognised by computers). Unpickling is the inverse where a byte stream is converted back into an object hierarchy. An example is a string of letters ‘hello’ being converted into the 1s and 0s that a computer recognises

16. Pychecker and Pylint perform static analysis that detect the bugs in code.

17. Arguments in Python are passed by reference. It means if the function changes what the argument is referencing, this change is reflected when calling the function.

So ,if the code is as follows:

student={‘Charlotte’:21,'Emily':19,’Holly’:12}

def test(student):

   new={'Joe':30,'Ryan':28}

   student.append(new)

   print("Inside the function",student)

   return

test(student)

print("outside the function:",student)

Then the output for ‘test(student)’ and ‘print("outside the function:",student)’ will be the same – {‘Charlotte’ : 21, ‘Emily’ : 19, ‘Holly’ : 12, ‘Joe’ : 30, ‘Ryan’ : 28}. The function changed how the data was outside of the function

18. Dictionary and List comprehensions are ways to make dictionaries (a unique way of storing data objects in key-value pairs) and lists (a collection of objects separated by commas in square brackets) in python respectively.

An example of list comprehension is:

list = [i for i in range(10)]

list = [0,1,2,3,4,5,6,7,8,9)

An example of dictionary comprehension is:

dictionary = dict([(i, i+10) for i in range(4)])

dictionary = {0:10, 1:11, 2:12, 3:13}

19. A namespace is a collection of currently defined names along with information about the object that each name references. E.g., if x = 12, then the namespace dictionary will hold the information about the object name (x) and the object value (12) that is references. It ensures that names can only correspond to one value and things won’t get mixed up.

20. Pass is a null operation and allows execution to continue at the next statement. This differs to return, which ends the operation. Pass instead just skips onto the next bit. It is sometimes used as a placeholder for future code.

21. Unit test is a way to test code in python. It allows small chunks to be individually tested, so the source of any error can be easily identified.

22. If you have a sequence of data, and want to work with a specific segment of it (opposing to the whole thing) or specific individual data in the sequence (e.g., the first and last value), then slicing can be used to extract the desired values from the sequence.

23. Negative indexing is used to slice a sequence backwards e.g., if you have a sequence of data and you want the last 3 values, rather than working out what positive value the final 3 correspond to, you can simply start from -1 (the last character) and slide backwards.

24. Ternary operators are conditional expressions, meaning that they work based on whether something is true or not. E.g.,:

nice = input(“Are you nice?y/n”)

is\_nice = nice == ‘y’

if is\_nice:

print(“The person is nice”)

else:

print(“The person is not nice”)

25. \*args and \*\*kwargs \* allow you to pass multiple arguments (args) or keyword arguments (kwargs) to a function. If you have made a function, often you have to define the arguments beforehand, and so in the future you are limited to passing through the number of arguments defined and it is not very flexible. E.g., if you have a function that adds up numbers, you may have to define 3 arguments – def addup(a, b, c). But in the future you might want to add several numbers using this function - \*args and \*\*kwargs can replace the arguments to allow you to add as many as you want.

26. Both range and xrange are used to iterate through a for-loop a certain number of times. Range is for python 3 and xrange is for python 2. They are different in many ways, one being that xrange is faster than range. Another difference is that operations such as slicing can be used in conjunction with range but not xrange.

27. Flask is a Python library (not already installed in python though) which can be used for making web applications.

28. In SQL, a clustered index defines the order in which data is physically stored in a table. A non-clustered index doesn’t sort the physical data in a table. E.g., if you are entering names and the clustered index sorts it in alphabetical order, then a new row with a name beginning with ‘A’ will not be put underneath the previous row, it will be immediately be put at the top row – a non-clustered index does not do this.

29. A deadlock is a situation that occurs in OS when any process enters in a waiting state because the demanded resource is being held by another waiting process.

30. A livelock, on the other hand, is almost similar to a deadlock, except that the states of the processes which are involved in a livelock always keep on changing to one another, none progressing.

**String methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Capitalize() | Puts a string in sentence case | string1 = 'hello how are you'  print(string1.capitalize())  output = Hello how are you |
| Casefold() | Makes all letters in a string lower case | string1 = 'HeLlO HoW arE You'  print(string1.casefold())  output = hello how are you |
| Center() | Makes a string take up a certain amount of space and places it in the middle | string1 = 'HeLlO HoW arE You'  print(string1.center(44))  output = HeLlO HoW arE You |
| Endswith() | Returns ‘true’ or ‘false’ on the condition that the string ends with the specified value | string1 = 'HeLlO HoW arE You'  print(string1.endswith('u'))  output = True |
| Count() | Counts the instances of the target value within the string | string1 = 'HeLlO HoW arE You'  print(string1.count('o'))  output = 2 |
| Find() | Returns the position in the string that the first instance of the target value occurs | string1 = 'HeLlO HoW arE You'  print(string1.find('H'))  output = 0 |
| Format() | Allows you to print the string and add new values at various points, denoted by a {} | string1 = 'Hello how are you' name = 'Charlotte'  print(string1+ ", {}".format(name))  output = Hello how are you, Charlotte |
| Index() | Returns the position of a character in a string (like find()). Index and find have only subtle differences, e.g., what is returned if the character/value is not found in the string | string1 = 'Hello how are you'  print(string1.index('o'))  output = 4 |
| Isalnum() | Returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9). False would be returned if any symbols or spaces are present | string1 = 'Hello how are you'  print(string1.isalnum())  output = False (because of the spaces) |
| Isalpha() | Returns True if all the characters are alphabet (a-z). False would be returned if any symbols or spaces or numbers are present | string1 = 'Hellohowareyou'  print(string1.isalpha())  output = False (because of the spaces) |
| Isdigit() | Returns True if all the characters are numbers (0-9). False would be returned if any symbols or spaces or letters are present | string1 = '1234567'  print(string1.isdigit())  output = True |
| Islower() | Lets you know if a string contains all lower case letters (returns True) or not (returns False) | string1 = 'asdFgh'  print(string1.islower())  output = False |
| Isnumeric() | Lets you know if a string contains all number (returns True) or not (returns False) | string1 = '23456'  print(string1.isnumeric())  Output = True |
| Isspace() | Lets you know if a string is just empty space (returns True) or not (returns False) | string1 = ' '  print(string1.isspace())  Output = True |
| Istitle() | Lets you know if a string is in Sentence Case Lile This (returns True) or not (returns False) | string1 = 'Hello There '  print(string1.istitle())  output = True |
| Isupper() | Lets you know if a string is ALL CAPITAL LETTERS (returns True) or not (returns False) | string1 = 'HIYA '  print(string1.isupper())  output = True |
| Join() | Places the entire first string between all of the character in the second string | string1 = '000000000' string2 = '111'  print(string1.join(string2))  output = 100000000010000000001 |
| Lower() | Prints out a desired string in all lowercase letters | string1 = 'HELLO'  print(string1.lower())  output = hello |
| Lstrip() | Removes all spaces at the beginning of a string | string1 = ' Hi'  print(string1.lstrip())  Output = Hi |
| Replace() | You can change the object value of an entire string or just elements of a string | string1 = 'Hi' string2 = 'replaced'  print(string1.replace(string1, string2))  output = replaced |
| Rsplit() | Splits a string into separate elements in a list based on a defined character to say where the string should be split | string1 = 'yellow! pink! green!'  print(string1.rsplit("!"))  Output = [‘yellow’, ‘pink’, ‘green’] |
| rstrip() | Gets rid of any white spaces at the end of a string (opposite to lstrip i.e., strip spaces on the left hand side) | string1 = 'Hi '  print("{}, how are you?".format(string1.rstrip()))  output = Hi, how are you? |
| Split() | Splits a string into elements of a list wherever it detects a space. Like rstrip but the default version where you don’t specify where to split, it is automatically done at the spaces | string1 = 'yellow! pink! green!'  print(string1.split())  Output = [‘yellow’, ‘pink’, ‘green’] |
| Splitlines() | Like split except it doesn’t split at spaces, it splits stirng into list elements whenever detecting ‘\n’ | string1 = 'hi there\n how are you?'  print(string1.splitlines())  Output = Hi there  How are you? |
| Startswith() | After specifying a target character, this method says whether the string starts with this character (True) or not (False) | string1 = 'hi there\n how are you?'  print(string1.startswith('h'))  Output = True |
| Strip() | Gets rid of any spaces at the beginning and end of a string (opposing to just at the beginning in lstrip, and just at the end in rstrip) | string1 = ' hi there how are you? '  print("x, {}, x".format(string1.strip()))  Output = x, hi how are you, x |
| Swapcase() | Turns all lower case letters into upper, and all upper into lower within the same sting | string1 = 'hi there HOW are you?'  print(string1.swapcase())  Output = HI THERE how ARE YOU |
| Title() | Turns a string into sentence case | string1 = 'hi there HOW are you?'  print(string1.title())  Output = Hi There How Are You |
| Upper() | Capitalises all letters in a string | string1 = 'hi there HOW are you?'  print(string1.upper())  Output = HI THERE HOW ARE YOU |

**List methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Append() | Adds new items to a list | students = ['harry', 'sally'] new\_student = 'charlotte' students.append(new\_student) print(students)  #output = ['harry', 'sally', 'charlotte'] |
| Clear() | Gets rid of all contents of a list | list = ['apples', 'banana', 'pear'] print(list.clear())  #output = None |
| Copy() | Returns a duplicate of the list | list = ['apples', 'banana', 'pear'] print(list.copy())  #output = ['apples', 'banana', 'pear'] |
| Count() | When given a desired object value, this function counts how many times the value is present in the list | list = ['apples', 'banana', 'pear'] print(list.count('apples'))  #output = 1 |
| Extend() | Adds a set of items in a list onto the end of an existing list | list = ['apples', 'banana', 'pear'] list2 = ['oranges', 'cherry'] list.extend(list2) print(list) |
| Index() | Tells you the position that a target list element is located at in a list | list = ['apples', 'banana', 'pear'] print(list.index('banana'))  #output = 1 |
| Insert() | Puts a new element at a specified location in a list | list = ['apples', 'banana', 'pear'] list.insert(1 ,'cherry') print(list)  #output = ['apples', 'cherry', 'banana', 'pear'] |
| Pop() | Removes an item at a pre-defined position in the list and tells you which item was removed | list = ['apples', 'banana', 'pear'] print(list.pop(1)) print(list)  #output = banana #['apples', 'pear'] |
| Remove() | If given a value to remove, this function removes all elements of it in the list | list = ['apples', 'banana', 'pear'] list.remove('pear') print(list) #output = ['apples', 'banana'] |
| Reverse() | Makes the order of items in the list reversed | list = ['apples', 'banana', 'pear'] list.reverse() print(list) #output = ['pear', 'banana', 'apples'] |
| Sort() | Returns the list in ascending order by default (but you can get it to sort the list according to other criteria too e.g., descending order) | list = ['apples', 'banana', 'pear', 'cherry', 'orange'] list.sort() print(list) #output = ['apples', 'banana', 'cherry', 'orange', 'pear'] |

**Tuple methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Count() | counts the occurrence of an element in the tuple. It returns the occurrence of the the element passed during call. It requires a parameter which is to be counted. It returns error if the parameter is missing. | data = (2,2,3,8,10) print(data.count(2))  #output = 2 |
| Index() | Returns the position of a given element in the tuple | data = (3,4,8,10) print(data.index(4))  #output = 1 |

**Dictionary methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Clear() | Removes all items from a dictionary | d = {1: "charlotte", 2: "sophie"} d.clear() print(d)  #output = {} |
| Copy() | Creates a copy of the dictionary | d = {1: "charlotte", 2: "sophie"} d.copy() print(d)  #output = {1: 'charlotte', 2: 'sophie'} |
| Fromkeys() | creates a new dictionary from the given sequence of elements with a value provided by the user. | keys = {'a', 'e', 'i', 'o', 'u' }  value = [1]  vowels = dict.fromkeys(keys, value)  print(vowels) |
| Get() | Returns the value associated with a specified key | d = {'name': 'sophie', 'age' : 22}  print('name', d.get('name'))  #output = name sophie |
| Items() | Returns the key-value pairs as pairsof tuples in a list | car = {  "brand": "Ford",  "model": "Mustang",  "year": 1964  }  x = car.items()  print(x)  output = dict\_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)]) |
| Keys() | Returns the keys of a dictionary as a list | d = {'name': 'sophie', 'age': 22}  print(d.keys())  #output = dict\_keys(['name', 'age']) |
| Pop() | When given a key, it removes the key-value pair from the dictionary | d = {'name': 'sophie', 'age': 22}  d.pop('name') print(d)  #output = {'age': 22} |
| Popitem() | Removes the last key/value pair in a dictionary | d = {'name': 'sophie', 'age': 22}  d.popitem() print(d)  #output = {'name': 'sophie'} |
| Setdefault() | Returns the value of the item corresponding to the specified key | d = {'name': 'sophie', 'age': 22}  x = d.setdefault('age') print(x)  #output = 22 |
| Update() | Changes key-value pairs in a dictionary with another specified key-value pair | d = {'name': 'sophie', 'age': 22} d1 = {'name': 'charlotte'}  d.update(d1) print(d)  #output = {'name': 'charlotte', 'age': 22} |
| Values() | Returns all the values in a dictionary | d = {'name': 'sophie', 'age': 22}  print(d.values())  #output = dict\_values(['sophie', 22]) |

**Set methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Add() | Adds another item to a set | letters = {'a', 'b', 'c', 'e'}  letters.add('d') print(letters)  #output = {'c', 'a', 'd', 'b', 'e'} |
| Clear() | Gets rid of all elements in a set | letters = {'a', 'b', 'c', 'e'}  letters.clear() print(letters)  #output = set() |
| Copy() | Returns a copy of the set | letters = {'a', 'b', 'c', 'e'}  letters.copy() print(letters)  #output = {'b', 'c', 'e', 'a'} |
| Difference() | Returns the items that are different between 2 sets | set1 = {1, 2, 5, 10} set2 = {1, 2, 3, 4}  print(set1.difference(set2))   #output = {10, 5} |
| Intersection() | Opposite of difference, in that it returns the items that are common between the 2 sets | x = {"apple", "banana", "cherry"}  y = {"google", "microsoft", "apple"}  z = x.intersection(y)  print(z) |
| Issubset() | Answers whether all the values of one set are present in another set, by returning True or False | x = {"a", "b", "c"} y = {"f", "e", "d", "c", "b", "a"}  print(x.issubset(y))  #output = True |
| Issuperset() | Reverse of the above, answers whether the items in a one set match all the values in another set | x = {"h", "j", "c"} y = {"f", "e", "d", "c", "b", "a"}  print(y.issuperset(x))  #output = False |
| Pop() | Gets rid of the first element in a set | x = {"h", "j", "c"}  x.pop() print(x)  #output = {'j', 'c'} |
| Remove() | Deletes a specified element from a set | x = {"h", "j", "c"}  x.remove('j') print(x)  #output = {'h', 'c'} |
| Symmetric\_difference() | When given 2 sets, this prints all of the elements that are unique to each set and are not found to be common across both sets | A = {'a', 'b', 'c', 'd'} B = {'c', 'd', 'e' }  print(A.symmetric\_difference(B))  #output = {'a', 'b', 'e'} |
| Union() | Joins 2 sets together but doesn’t repeat duplicates | A = {'a', 'b', 'c', 'd'} B = {'c', 'd', 'e' }  print(A.union(B))  #output = {'b', 'c', 'a', 'e', 'd'} |
| Update() | Updates the set with new items, but returns None | A = {'a', 'b', 'c', 'd'} B = {'c', 'd', 'e' }  print(A.update(B))  #output = None |

**File methods**

|  |  |  |
| --- | --- | --- |
| Method | Description | Example |
| Read() | Reads content of a file | with open('todo.txt', 'r') as todo\_file:  todo = todo\_file.read()  # print(*todo)*  output = content of ‘todo.txt’ |
| Readline() | Returns the first line of a file | f = open("demofile.txt", "r") print(f.readline())  output = first line of ‘demofile.txt’ |
| Readlines() | returns a list containing each line in the file as a list item | f = open("demofile.txt", "r")  print(f.readlines())  output = [‘hello’, ‘how are you?’] |
| Write() | Allows new content to be written into a file | With open(‘new\_file.txt.’, ‘w’) as new\_file:  new\_file.write(“HI!”) |
| Writelines() | Writes new content into a file in a more structures way in lines, needs list input | With open(‘new\_file.txt.’, ‘w’) as new\_file:  new\_file.writelines(["See you soon!", "Goodbye."]) |