# THEORY QUESTIONS ASSIGNMENT

Python based theory

**1. Python theory questions 30 points**

1. What is Python and what are its main features?
Python is one of many programming languages. It's very often recommended as a first language to learn, as it is relatively easy compared with other ones. This is one of its main features, others include it being object-oriented (creates objects that contain data and functions), its portability (can be run on various platforms), it's interpreted, so no need to compile the code, it is free and open source, it's a high-level language (which is easier to use for humans, as is closer to the natural language), it's extensible (Python code can be written into C or C++) and it's dynamically typed (type of variable is checked during run-time).

2. Discuss the difference between Python 2 and Python 3
In Python 2 'print' is a statement, in Python 3 – a function. Strings are stored as ASCII in Python 2, whilst in Python 3 they are stores as UNICODE. Python 2 syntax is more complicated than in Python 3. Python 3 is used in a lot of fields (like Data Science, Software Engineering etc.), whilst Python 2 was mostly used to become a DevOps Engineer. In Python 2, when dividing two integers, we get an integral value – in Python 3, a floating-point value. For iterations, Python 2 uses xrange(), and Python 3, range(). It's possible to port Python 2 code to Python 3, but Python 3 is not backward compatible with Python 2. The values of global variables change inside a for-loop in Python 2, but they don't change in Python 3. Exceptions in Python 2 are enclosed in notations, and in Python 3 in parentheses.
This is not an exhaustive list, but it presents the most well-known differences.

3. What is PEP 8?
PEP 8 is a Style Guide for Python Code, a document that contains guidelines and showcases best practices on how to write Python code.

4. In computing / computer science what is a program?
It is a set of instructions for a computer to perform.

5. In computing / computer science what is a process?
It is an instance of a program that is running on a computer.

6. In computing / computer science what is cache?
It is a small amount of memory that is used to hold instructions and data likely to be reused, so they can be easily accessed and fetched faster. They are stored there temporarily.

7. In computing / computer science what is a thread and what do we mean by multithreading?
A thread is a small set of instructions that are executed and scheduled by central processing unit independently of a parent process. Multithreading is the ability of operating system or a program to provide multiple threads to run simultaneously.

8. In computing / computer science what is concurrency and parallelism and what are the differences?
Concurrency means that more than one task is being executed in an overlapping time period. Parallelism means that the tasks are processed in parallel at the exact same time. To achieve parallelism there must be multiple threads on separate CPUs/CPU cores etc., whilst concurrency can be achieved on a single core. Concurrency is about managing multiple tasks, whilst parallelism is about running them at the same time.

9. What is GIL in Python and how does it work?
GIL (Global Interpreter Lock) is a lock (also known as mutex) that allows only one thread to be in a state of execution at any point in time. This is to prevent a race condition (when the behaviour of the code or system relies on the sequence of execution defined by uncontrollable events). GIL makes sure only one thread is running, therefore diminishing this possibility. This might however cause limitations on the use of multi-core systems in the context of threading.

10. What do these software development principles mean: DRY, KISS, BDUF
DRY = don’t repeat yourself, recommendation aimed at reducing the repetition of information.
KISS = keep it simple, stupid, which recommends avoiding unnecessary complexity in systems.
BDUF = big design up front, that is recommendation for the design to be perfected and completed up-front, before its implementation has been initialised.

11. What is a Garbage Collector in Python and how does it work?
Garbage Collector is a process in Python that deletes unwanted objects automatically to free the memory space. It works periodically, freeing and reclaiming blocks of memory no longer in use. It runs during program execution and when an object’s reference count reaches zero, it is being triggered (it reaches zero when it’s deleted, reassigned or its reference goes out of scope).

12. How is memory managed in Python?
Memory in Python is managed automatically. Reclaiming of space is mostly handled by reference counting, which keeps track of number of references referring to an object, and if it reaches zero – Garbage Collector is triggered. There are two parts: stack memory and heap memory. In the stack memory Python stores methods and method calls, in the heap memory – objects and data structures.

13. What is a Python module?
Module is a file that contains Python definitions and statements. Modules are used to break down sizeable programs, as it is more manageable and easier to organize small files. Modules can be built-in and user defined.

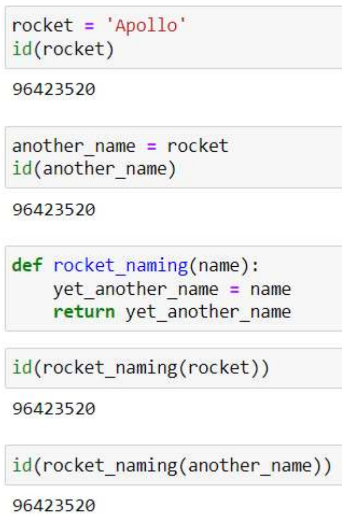
14. What is docstring in Python?
It is a string literal that appears after the definition of a function, class, module or method. It is used to document the code (allowing programmers to understand what it

does without the need to read the details of implementation), and it is contained within triple quotation marks.

15. What is pickling and unpickling in Python? Example usage.
Pickling is a process that converts Python objects into byte streams. Unpickling is the reverse process. It is used to allow storing of the code in a file or a database (for example for saving trained state of neural network after the training phase, so that it doesn't have to be retrained) or to transport data over network.

16. What are the tools that help to find bugs or perform static analysis?
PyChecker is an opensource static analysis tool that finds bugs in Python. It can find things like unused globals and locals; passing the wrong number of parameters to function; using class methods and attributes that don't exist; using a variable before assigning it, detects missing docstrings etc. Pylint is another static code analyser. It checks the code without the need to run it. It scans the code for errors, ensures it follows PEP8 recommendations and can make suggestions how to improve it. There is also Pyflakes that checks files for errors, works by parsing the source file (so does not execute the modules to check them). It doesn't perform any checks on style (this can be achieved by using flake8 instead, it combines Pyflakes with PEP8 recommendations, with ability to configure it on a project basis).

17. How are arguments passed in Python by value or by reference? Give an example.
Arguments in Python are passed neither by value nor by reference, but rather by assignment. Whenever a function is called, each parameter of that function is assigned to the object they were passed in. We can show it by checking the identity number of objects – it stays constant, even if we assign it a different name:



```
rocket = 'Apollo'
id(rocket)
```

```
96423520
```

```
another_name = rocket
id(another_name)
```

```
96423520
```

```
def rocket_naming(name):
    yet_another_name = name
    return yet_another_name
```

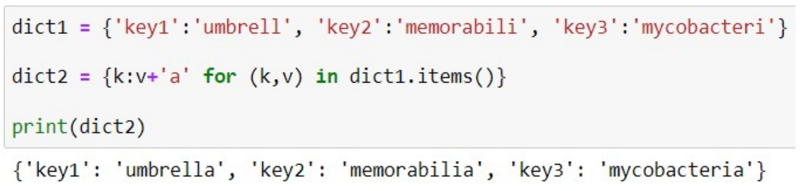
```
id(rocket_naming(rocket))
```

```
96423520
```

```
id(rocket_naming(another_name))
```

```
96423520
```

18. What are Dictionary and List comprehensions in Python? Provide examples.
Dictionary comprehension in Python is a method offering shorter syntax to change one dictionary into another, with possibility of original items being conditionally transformed to fit desired criteria. For example:



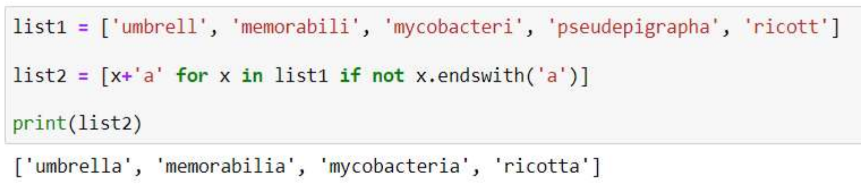
```
dict1 = {'key1':'umbrell', 'key2':'memorabili', 'key3':'mycobacteri'}

dict2 = {k:v+'a' for (k,v) in dict1.items()}

print(dict2)
```

```
{'key1': 'umbrella', 'key2': 'memorabilia', 'key3': 'mycobacteria'}
```

This is true as well for the list comprehension:



```
list1 = ['umbrell', 'memorabili', 'mycobacteri', 'pseudepigrapha', 'ricott']

list2 = [x+'a' for x in list1 if not x.endswith('a')]

print(list2)
```

```
['umbrella', 'memorabilia', 'mycobacteria', 'ricotta']
```

19. What is namespace in Python?
Namespace is a system of mapping name to an object, implemented as a dictionary. There can be different namespaces in existence at a specific time, but they will be isolated, meaning that even if the same name exists in different modules, they won't collide. There can be: built-in namespace (containing all built-in objects' names), global namespace (defined in the body of the program) and local namespace (existing within the function).

20. What is pass in Python?
Pass is a statement that results in no operation when it is executed. It allows execution to continue at the next statement. It can be useful as a placeholder when constructing new code or for example to temporarily disable parts of statements or functions to test the rest of the code.

21. What is unit test in Python?
Unit testing is a method of testing the source code, in which small units of code are tested separately (by unit tests), to ensure they perform as expected. This is helpful when changing the code (to make sure that existing functionally in other parts has not been impacted), and it also makes isolating errors easier.

22. In Python what is slicing?
Slicing in Python means extracting parts of a string, list or a tuple. By stating indices, it can extract specific range of elements. Syntax for it is object[start:stop:step].

23. What is a negative index in Python?
Negative index is an option in Python to access an object from its end. It starts at -1. For example, if we wanted to access the last element of a list, we would do it by calling example_list[-1]. It can be also used in slicing.

24. How can the ternary operators be used in Python? Give an example.
Ternary operators (aka conditional expressions), consisting of three operands, evaluate statement based on a condition being True or False. The syntax is: when_true if condition else when_false. It's great for replacing simple if statements with one line of code. For example:

```python
rocket_name = 'Apollo 11'
landed = True if rocket_name == 'Apollo 11' else False

print(f'{rocket_name} landed on the Moon: {landed}')
```

```
Apollo 11 landed on the Moon: True
```

```python
rocket_name = 'Apollo 8'
landed = True if rocket_name == 'Apollo 11' else False

print(f'{rocket_name} landed on the Moon: {landed}')
```

```
Apollo 8 landed on the Moon: False
```

25. What does this mean: *args, **kwargs? And why would we use it?
*args, **kwargs allow to pass multiple arguments or keyword (named) arguments to a function. They are useful in writing a function where we won't know the number of arguments until later.
For example:

```python
def sample_func(*args):
    total_sum = 0
    for arg in args:
        total_sum += arg
    return total_sum
```

```python
sample_func(2)
```

```
2
```

```python
sample_func(1,3,5)
```

```
9
```

```python
sample_func(3,3,4,5,6,72,2)
```

```
95
```

And for kwargs:

```python
def sample_keyargs(**kwargs):
    total_sum = 0
    for key,value in kwargs.items():
        print(f'Key name: {key}')
        print(f'Adding {value} to total_sum.')
        total_sum += value
        print(f'Total sum is now: {total_sum}')
```

```python
sample_keyargs(a=1)
```

```
Key name: a
Adding 1 to total_sum.
Total sum is now: 1
```

```python
sample_keyargs(a=1,b=3,c=5)
```

```
Key name: a
Adding 1 to total_sum.
Total sum is now: 1
Key name: b
Adding 3 to total_sum.
Total sum is now: 4
Key name: c
Adding 5 to total_sum.
Total sum is now: 9
```

26. How are range and xrange different from one another?
range() returns a list object, whilst xrange() returns generator object – it will create values only when they are needed. Because of that, xrange() object will always take the same amount of memory (less than a range() object), but the range() will be faster to iterate over than xrange().

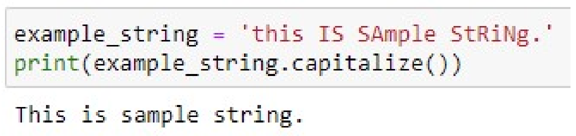
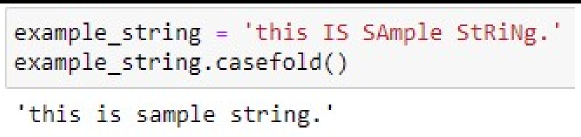
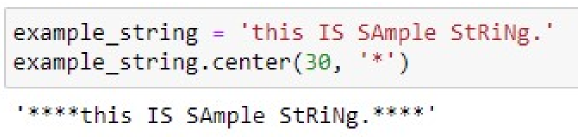
27. What is Flask and what can we use it for?
It is a micro-web framework written in Python. We can use it for building web-based applications.

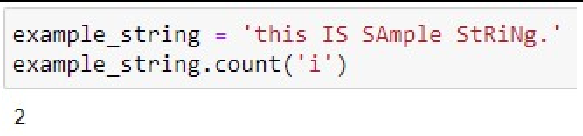
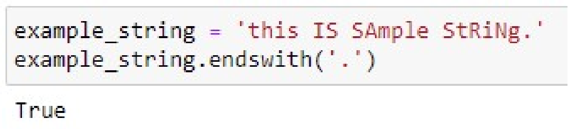
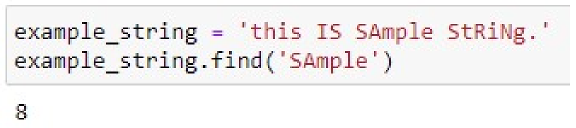
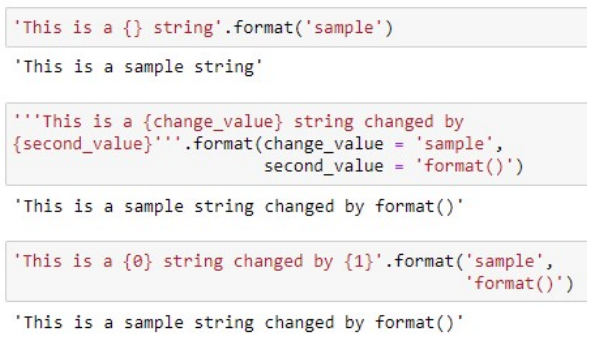
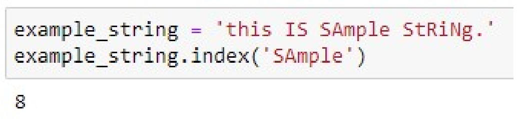
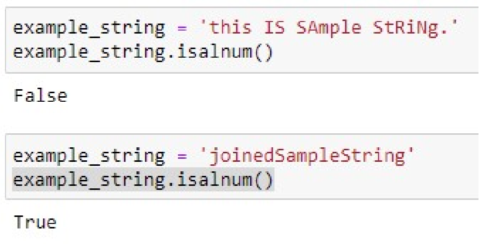
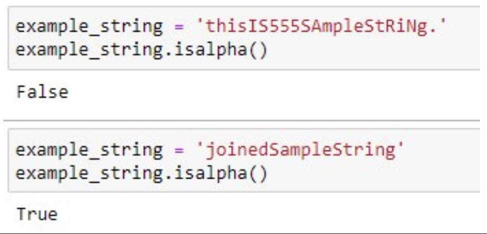
28. What are clustered and non-clustered index in a relational database?
Clustered index is an index that creates a physical order based on chosen columns and is then it is used to store the data that way. There can be only one clustered index (for example primary key). Non-clustered index doesn't sort the data, it only holds references to it and is stored independently from it. There can be multiple non-clustered indexes.

29. What is a 'deadlock' a relational database?
Deadlock happens when two or more transactions wait for the other to release locks on required resources, and they keep waiting in a state of no action.

30. What is a 'livelock' a relational database?
Livelock happens when, because of overlapping shared locks, two or more transactions are constantly denied exclusive lock on required resources. Despite that, they continue to work, but because of lack of that exclusive lock, they are never able to complete the task, and that prevents other transactions to access resources they are holding a lock on.

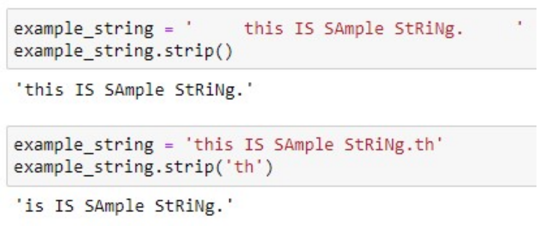
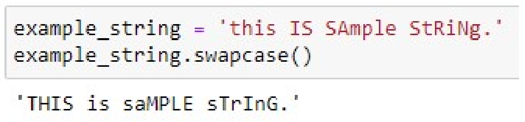
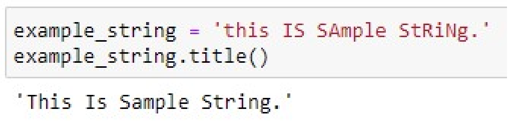
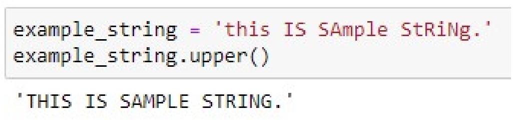
**2. Python string methods: describe each method and provide an example 29 points**

| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **capitalize()** | Returns a string with first letter capitalized and the rest lower case. |  `example_string = 'this IS SAmple StRiNg.'`<br>`print(example_string.capitalize())`<br>`This is sample string.` |
| **casefold()** | Returns a string with all letters lower case (it's stronger than lower(), as it will convert even things like German ß into 'ss'). |  `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.casefold()`<br>`'this is sample string.'` |
| **center()** | Returns a centered string using chosen special character. First (length) parameter is required, second (character) is optional (space is a default). |  `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.center(30, '*')`<br>`'****this IS SAmple StRiNg.****'` |

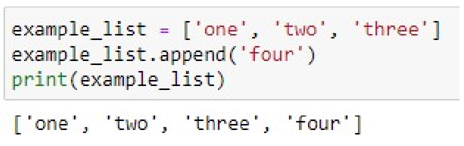
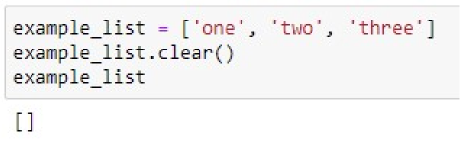
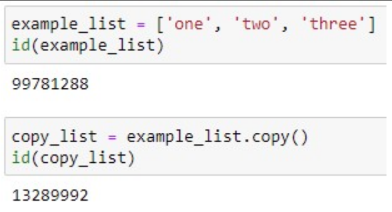
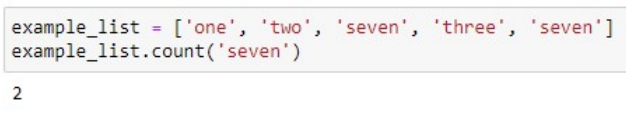
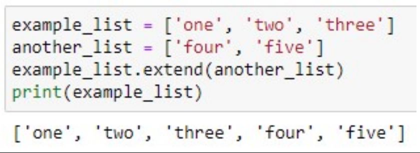
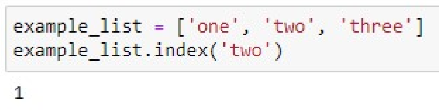
|  |  |  |
|---|---|---|
| **count()** | Returns number of times specified value appears in a string. It has value, start and stop parameters (the last two optional). | <br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.count('i')`<br>`2` |
| **endswith()** | Returns true if a string ends with specified value (else false). Parameters are value, start, stop (the last two optional). | <br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.endswith('.')`<br>`True` |
| **find()** | Returns an integer value – if the substring is found, it will show the index position of the first occurrence of substring, if it doesn't exist, it returns -1. It has three parameters, value (required), start and end (optional). | <br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.find('SAmple')`<br>`8` |
| **format()** | Returns formatted string, with specified values formatted and inserted inside the string's placeholder, which are marked with {}. | <br>`'This is a {} string'.format('sample')`<br>`'This is a sample string'`<br>`'''This is a {change_value} string changed by`<br>`{second_value}'''.format(change_value = 'sample',`<br>`                        second_value = 'format()')`<br>`'This is a sample string changed by format()'`<br>`'This is a {0} string changed by {1}'.format('sample',`<br>`                                            'format()')`<br>`'This is a sample string changed by format()'` |
| **index()** | Returns the index of a substring if found, will raise an exception of not. It has three parameters, value as a required one, and start and stop optional. | <br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.index('SAmple')`<br>`8` |
| **isalnum()** | Returns True if all characters in a string are alphanumeric, False if some are not. | <br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.isalnum()`<br>`False`<br>`example_string = 'joinedSampleString'`<br>`example_string.isalnum()`<br>`True` |
| **isalpha()** | Returns True if all characters are alphabet letters, False if not. | <br>`example_string = 'thisIS555SAmpleStRiNg.'`<br>`example_string.isalpha()`<br>`False`<br>`example_string = 'joinedSampleString'`<br>`example_string.isalpha()`<br>`True` |

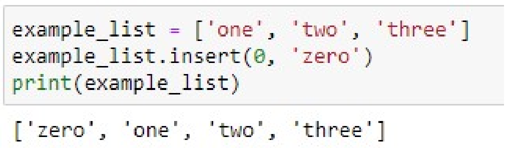
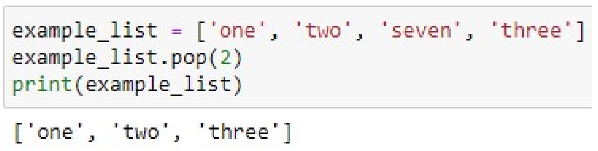
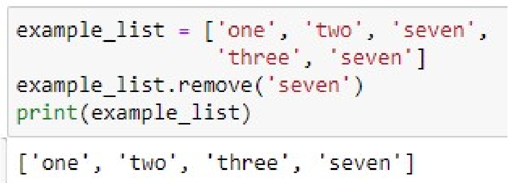
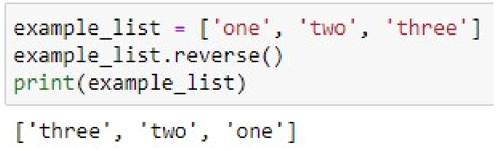
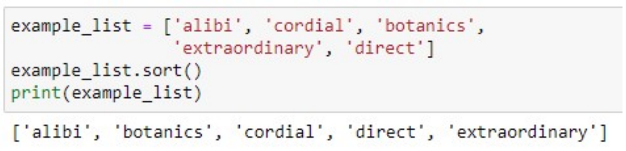
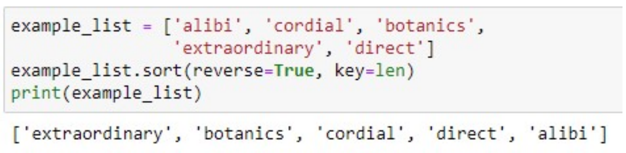
| | | |
|---|---|---|
| **isdigit()** | Returns True if all characters are digits, False if not. | `example_string = 'thisIS555SAmpleStRiNg.'`<br>`example_string.isdigit()`<br>`False`<br><br>`example_string = '77363839202'`<br>`example_string.isdigit()`<br>`True` |
| **islower()** | Returns True if all characters are lowercase, False if not. | `example_string = 'thisIS555SAmpleStRiNg.'`<br>`example_string.islower()`<br>`False`<br><br>`example_string = 'this is a sample string.'`<br>`example_string.islower()`<br>`True` |
| **isnumeric()** | Returns True if all characters are numeric, False if not. They can be also written in Unicode. | `example_string = '\u00BE'`<br>`example_string.isnumeric()`<br>`True`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.isnumeric()`<br>`False` |
| **isspace()** | Returns True if all characters are whitespace, False if not. | `example_string = '                    '`<br>`example_string.isspace()`<br>`True`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.isspace()`<br>`False` |
| **istitle()** | Returns True if each word in a string starts with uppercase, False if not. | `example_string = 'This Is Sample String.'`<br>`example_string.istitle()`<br>`True`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.istitle()`<br>`False` |
| **isupper()** | Returns True if all characters are uppercase, False if not. | `example_string = 'THIS IS SAMPLE STRING'`<br>`example_string.isupper()`<br>`True`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.isupper()`<br>`False` |
| **join()** | Returns a string with all elements of an iterable joined with specified character. | `example_list = ['This', 'is', 'sample', 'iterable.']`<br>`print(' '.join(example_list))`<br>`This is sample iterable.` |
| **lower()** | Converts all uppercase characters into lowercase. | `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.lower()`<br>`'this is sample string.'` |

| | | |
|---|---|---|
| **lstrip()** | Removes characters from the left (can be specified what characters, space is default to remove) | `example_string = '    this IS SAmple StRiNg.'`<br>`example_string.lstrip()`<br>`'this IS SAmple StRiNg.'`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.lstrip('this ')`<br>`'IS SAmple StRiNg.'` |
| **replace()** | Replaces all the occurrences of defined value with another value (can be limited to specified number with third, optional, parameter) | `example_string = 'Lig Llue Lalloon'`<br>`example_string.replace('L', 'B')`<br>`'Big Blue Balloon'`<br><br>`example_string = 'Lig Llue Lalloon'`<br>`example_string.replace('L', 'B', 2)`<br>`'Big Blue Lalloon'` |
| **rsplit()** | Splits string into a list starting from right, with optional values of separator (default: space) and maxsplit (default: all). | `example_string = 'One, two, three'`<br>`example_string.rsplit(',')`<br>`['One', ' two', ' three']`<br><br>`example_string = 'One, two, three'`<br>`example_string.rsplit(',', 1)`<br>`['One, two', ' three']` |
| **rstrip()** | Removes characters from the right (can be specified what characters, space is default to remove). | `example_string = 'this IS SAmple StRiNg.     '`<br>`example_string.rstrip()`<br>`'this IS SAmple StRiNg.'`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.rstrip(' StRiNg.')`<br>`'this IS SAmple'` |
| **split()** | Splits string into a list (starting from left), with optional values of separator (default: space) and maxsplit (default: all). | `example_string = 'One, two, three'`<br>`example_string.split(',')`<br>`['One', ' two', ' three']`<br><br>`example_string = 'One, two, three'`<br>`example_string.split(',', 1)`<br>`['One', ' two, three']` |
| **splitlines()** | Splits string into a list on line breaks. Line breaks can be kept or not (default: False) | `example_string = "this IS SAmple StRiNg.\nSo much SamplE."`<br>`example_string.splitlines()`<br>`['this IS SAmple StRiNg.', 'So much SamplE.']`<br><br>`example_string = "this IS SAmple StRiNg.\nSo much SamplE."`<br>`example_string.splitlines(True)`<br>`['this IS SAmple StRiNg.\n', 'So much SamplE.']` |
| **startswith()** | Returns True if string starts with specified character, False if otherwise. | `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.startswith('t')`<br>`True`<br><br>`example_string = 'this IS SAmple StRiNg.'`<br>`example_string.startswith('G')`<br>`False` |

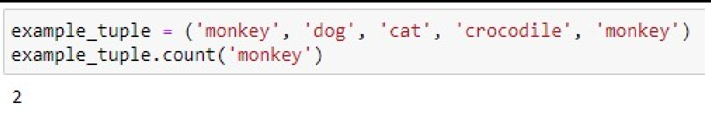
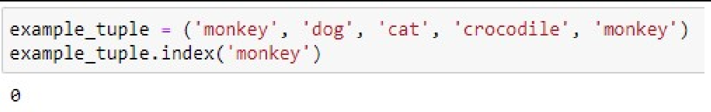
| | | |
|---|---|---|
| **strip()** | Removes characters from the beginning and end of string (can be specified what characters, space is default to remove). |  `example_string = '    this IS SAmple StRiNg.    '`<br>`example_string.strip()`<br>`'this IS SAmple StRiNg.'`<br>`example_string = 'this IS SAmple StRiNg.th'`<br>`example_string.strip('th')`<br>`'is IS SAmple StRiNg.'` |
| **swapcase()** | Swaps uppercase letters to lower-, and lower- to up-percase. |  `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.swapcase()`<br>`'THIS is saMPLE sTrInG.'` |
| **title()** | Capitalizes first letters of each word, makes all the rest of characters lower-case. |  `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.title()`<br>`'This Is Sample String.'` |
| **upper()** | Changes all characters to uppercase. |  `example_string = 'this IS SAmple StRiNg.'`<br>`example_string.upper()`<br>`'THIS IS SAMPLE STRING.'` |

**3. Python list methods: describe each method and provide an example 11 points**

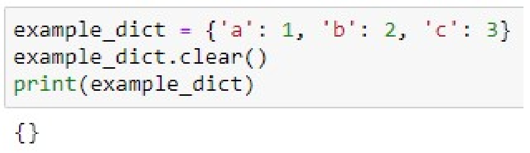
| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **append()** | Adds element at the beginning of the list. |  `example_list = ['one', 'two', 'three']`<br>`example_list.append('four')`<br>`print(example_list)`<br>`['one', 'two', 'three', 'four']` |
| **clear()** | Clears the list (removes all el-ements). |  `example_list = ['one', 'two', 'three']`<br>`example_list.clear()`<br>`example_list`<br>`[]` |
| **copy()** | Returns copy of the list. |  `example_list = ['one', 'two', 'three']`<br>`id(example_list)`<br>`99781288`<br>`copy_list = example_list.copy()`<br>`id(copy_list)`<br>`13289992` |
| **count()** | Returns number of occur-rences of specified value. |  `example_list = ['one', 'two', 'seven', 'three', 'seven']`<br>`example_list.count('seven')`<br>`2` |
| **extend()** | Adds all elements of iterable to the end of the list. |  `example_list = ['one', 'two', 'three']`<br>`another_list = ['four', 'five']`<br>`example_list.extend(another_list)`<br>`print(example_list)`<br>`['one', 'two', 'three', 'four', 'five']` |
| **index()** | Returns index of first occur-rence of specified value. |  `example_list = ['one', 'two', 'three']`<br>`example_list.index('two')`<br>`1` |

| | | |
|---|---|---|
| **insert()** | Inserts value at a specified position. |  `example_list = ['one', 'two', 'three']`<br>`example_list.insert(0, 'zero')`<br>`print(example_list)`<br>`['zero', 'one', 'two', 'three']` |
| **pop()** | Removes value at a specified position. Default is last item (-1). |  `example_list = ['one', 'two', 'seven', 'three']`<br>`example_list.pop(2)`<br>`print(example_list)`<br>`['one', 'two', 'three']` |
| **remove()** | Removes the first occurrence of specified value. |  `example_list = ['one', 'two', 'seven',`<br>`                'three', 'seven']`<br>`example_list.remove('seven')`<br>`print(example_list)`<br>`['one', 'two', 'three', 'seven']` |
| **reverse()** | Reverses sorting order of elements. |  `example_list = ['one', 'two', 'three']`<br>`example_list.reverse()`<br>`print(example_list)`<br>`['three', 'two', 'one']` |
| **sort()** | Sorts the list, it's ascending by default, there is optional parameter of defining a key. |  `example_list = ['alibi', 'cordial', 'botanics',`<br>`                'extraordinary', 'direct']`<br>`example_list.sort()`<br>`print(example_list)`<br>`['alibi', 'botanics', 'cordial', 'direct', 'extraordinary']`<br> `example_list = ['alibi', 'cordial', 'botanics',`<br>`                'extraordinary', 'direct']`<br>`example_list.sort(reverse=True, key=len)`<br>`print(example_list)`<br>`['extraordinary', 'botanics', 'cordial', 'direct', 'alibi']` |

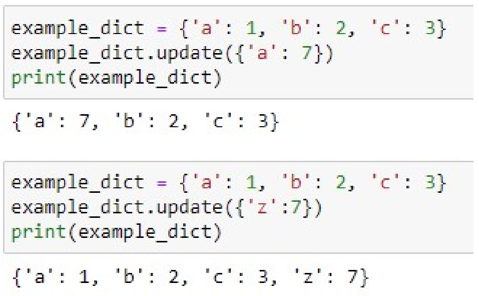
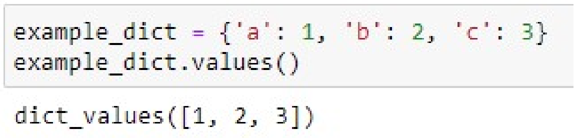
**4. Python tuple methods: describe each method and provide an example 2 points**

| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **count()** | Returns number of occurrences of specified value. |  `example_tuple = ('monkey', 'dog', 'cat', 'crocodile', 'monkey')`<br>`example_tuple.count('monkey')`<br>`2` |
| **index()** | Returns index of first occurrence of specified value. |  `example_tuple = ('monkey', 'dog', 'cat', 'crocodile', 'monkey')`<br>`example_tuple.index('monkey')`<br>`0` |

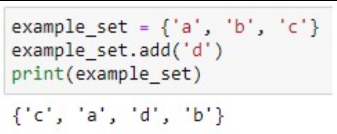
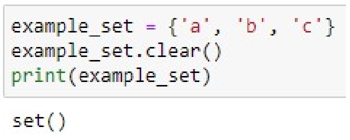
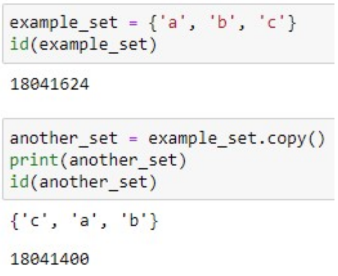
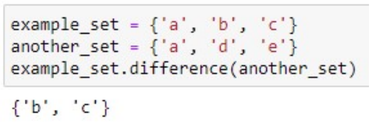
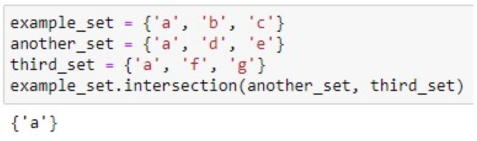
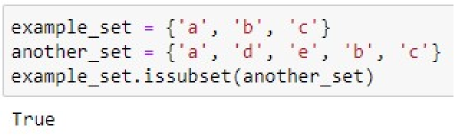
**5. Python dictionary methods: describe each method and provide an example 11 points**

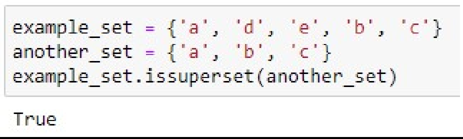
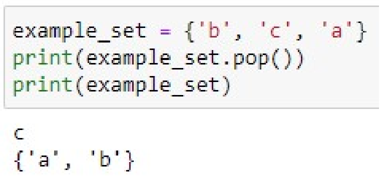
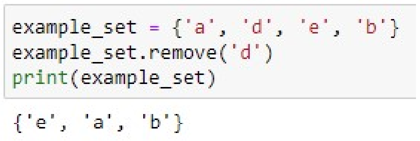
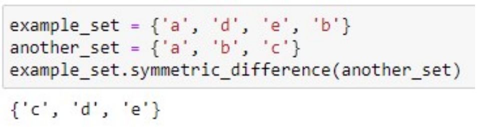
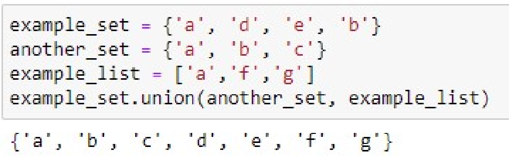
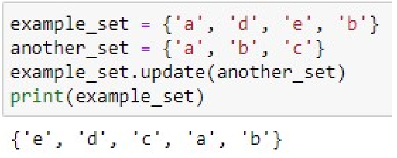
| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **clear()** | Removes all elements from the dictionary. |  `example_dict = {'a': 1, 'b': 2, 'c': 3}`<br>`example_dict.clear()`<br>`print(example_dict)`<br>`{}` |

| | | |
|---|---|---|
| **copy()** | Returns copy of dictionary. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `id(example_dict)` → `104873728` ; `another_dict = example_dict.copy()` `id(another_dict)` → `104895824` |
| **fromkeys()** | Returns a dictionary with specified keys and values. Keys parameter is required, value is optional (it will default to None if not specified). | `example_keys = ['The meaning of life is: ',` `'It also is: ', 'But seriously, it is: ']` `example_value = 42` `example_dictionary = dict.fromkeys(example_keys, example_value)` `print(example_dictionary)` → `{'The meaning of life is: ': 42, 'It also is: ': 42, 'But seriously, it is: ': 42}` |
| **get()** | Returns the value of specified key. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `example_dict.get('b')` → `2` |
| **items()** | Returns view of a dictionary with key-value pairs as tuples in a list. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `view_dictionary = example_dict.items()` `print(view_dictionary)` → `dict_items([('a', 1), ('b', 2), ('c', 3)])` |
| **keys()** | Returns dictionary keys in a list. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `example_dict.keys()` → `dict_keys(['a', 'b', 'c'])` |
| **pop()** | Removes specified element from dictionary. Parameter of keyname is required. We can also specify defaultvalue – in case the key we named doesn't exist, and we don't want to raise an error, we can return defaultvalue instead. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `example_dict.pop('a')` `print(example_dict)` → `{'b': 2, 'c': 3}` ; `example_dict = {'a': 1, 'b': 2, 'c': 3}` `value = example_dict.pop('z', 'This key does not exist.')` `print(value)` → `This key does not exist.` |
| **popitem()** | Removest the last inserted element of a dictionary. The removed element will be returned value of popitem() method, and will be presented as a tuple. | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `print(example_dict.popitem())` `print(example_dict)` → `('c', 3)` `{'a': 1, 'b': 2}` |
| **setdefault()** | Returns the value of the specified key. If key doesn't exist, it will insert it, with optional parameter of value (default is None). | `example_dict = {'a': 1, 'b': 2, 'c': 3}` `example_dict.setdefault('a', 4)` → `1` ; `example_dict = {'a': 1, 'b': 2, 'c': 3}` `example_dict.setdefault('z', 7)` `print(example_dict)` → `{'a': 1, 'b': 2, 'c': 3, 'z': 7}` |

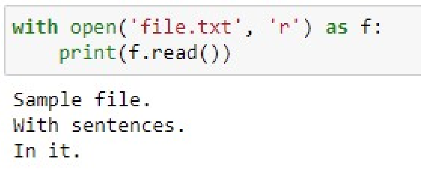
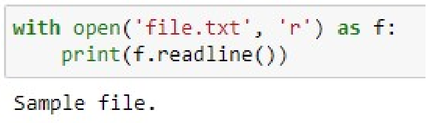
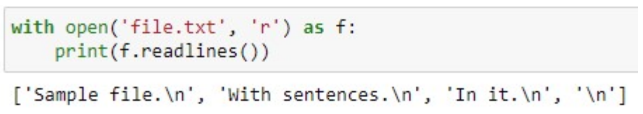
| | | |
|---|---|---|
| **update()** | Updates dictionary with specified elements – it updates existing ones or adds them if they don't exist. |  `example_dict = {'a': 1, 'b': 2, 'c': 3}`<br>`example_dict.update({'a': 7})`<br>`print(example_dict)`<br>`{'a': 7, 'b': 2, 'c': 3}`<br><br>`example_dict = {'a': 1, 'b': 2, 'c': 3}`<br>`example_dict.update({'z':7})`<br>`print(example_dict)`<br>`{'a': 1, 'b': 2, 'c': 3, 'z': 7}` |
| **values()** | Returns values of a dictionary as a list. |  `example_dict = {'a': 1, 'b': 2, 'c': 3}`<br>`example_dict.values()`<br>`dict_values([1, 2, 3])` |

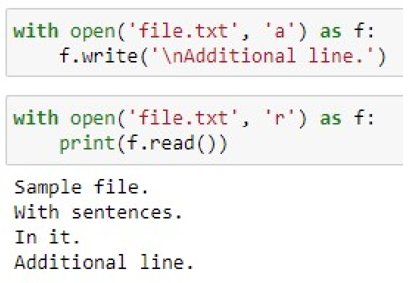
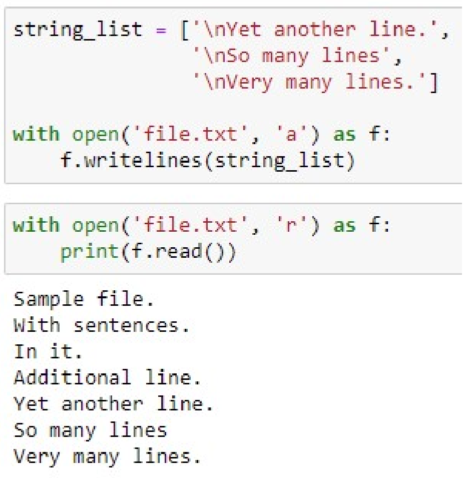
## 6. Python set methods: describe each method and provide an example 12 points

| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **add()** | Adds defined element to a set. |  `example_set = {'a', 'b', 'c'}`<br>`example_set.add('d')`<br>`print(example_set)`<br>`{'c', 'a', 'd', 'b'}` |
| **clear()** | Removes all elements from set. |  `example_set = {'a', 'b', 'c'}`<br>`example_set.clear()`<br>`print(example_set)`<br>`set()` |
| **copy()** | Returns copy of set. |  `example_set = {'a', 'b', 'c'}`<br>`id(example_set)`<br>`18041624`<br><br>`another_set = example_set.copy()`<br>`print(another_set)`<br>`id(another_set)`<br>`{'c', 'a', 'b'}`<br>`18041400` |
| **difference()** | Returns items that are unique to the first set in relation to the second set passed as a parameter. |  `example_set = {'a', 'b', 'c'}`<br>`another_set = {'a', 'd', 'e'}`<br>`example_set.difference(another_set)`<br>`{'b', 'c'}` |
| **intersection()** | Returns a set of common values of the compared sets. We can check more than two sets. |  `example_set = {'a', 'b', 'c'}`<br>`another_set = {'a', 'd', 'e'}`<br>`third_set = {'a', 'f', 'g'}`<br>`example_set.intersection(another_set, third_set)`<br>`{'a'}` |
| **issubset()** | Returns True if a set is a subset of another set (all values of the first set are present in the second set). |  `example_set = {'a', 'b', 'c'}`<br>`another_set = {'a', 'd', 'e', 'b', 'c'}`<br>`example_set.issubset(another_set)`<br>`True` |

| | | |
|---|---|---|
| **issuperset()** | Returns True if all values of the second set are present in the first set. |  `example_set = {'a', 'd', 'e', 'b', 'c'}`<br>`another_set = {'a', 'b', 'c'}`<br>`example_set.issuperset(another_set)`<br><br>`True` |
| **pop()** | Removes random item from set. |  `example_set = {'b', 'c', 'a'}`<br>`print(example_set.pop())`<br>`print(example_set)`<br><br>`c`<br>`{'a', 'b'}` |
| **remove()** | Removes specified element. |  `example_set = {'a', 'd', 'e', 'b'}`<br>`example_set.remove('d')`<br>`print(example_set)`<br><br>`{'e', 'a', 'b'}` |
| **symmetric_difference()** | Returns a set containing values that are not repeated in both sets. |  `example_set = {'a', 'd', 'e', 'b'}`<br>`another_set = {'a', 'b', 'c'}`<br>`example_set.symmetric_difference(another_set)`<br><br>`{'c', 'd', 'e'}` |
| **union()** | Returns a set containing values from all the sets (or iterables). It deduplicates repeated values and adds just one instance. |  `example_set = {'a', 'd', 'e', 'b'}`<br>`another_set = {'a', 'b', 'c'}`<br>`example_list = ['a','f','g']`<br>`example_set.union(another_set, example_list)`<br><br>`{'a', 'b', 'c', 'd', 'e', 'f', 'g'}` |
| **update()** | Updates set with values from another set, adds just one instance if value is present multiple times. |  `example_set = {'a', 'd', 'e', 'b'}`<br>`another_set = {'a', 'b', 'c'}`<br>`example_set.update(another_set)`<br>`print(example_set)`<br><br>`{'e', 'd', 'c', 'a', 'b'}` |

**7. Python file methods: describe each method and provide an example 5 points**

| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| **read()** | Returns the content of the file. |  `with open('file.txt', 'r') as f:`<br>`    print(f.read())`<br><br>`Sample file.`<br>`With sentences.`<br>`In it.` |
| **readline()** | Returns one line of the file. |  `with open('file.txt', 'r') as f:`<br>`    print(f.readline())`<br><br>`Sample file.` |
| **readlines()** | Returns all lines from the file as a list. |  `with open('file.txt', 'r') as f:`<br>`    print(f.readlines())`<br><br>`['Sample file.\n', 'With sentences.\n', 'In it.\n', '\n']` |

| | | |
|---|---|---|
| **write()** | Writes specified text to a file. |  `with open('file.txt', 'a') as f:`<br>`    f.write('\nAdditional line.')`<br><br>`with open('file.txt', 'r') as f:`<br>`    print(f.read())`<br><br>`Sample file.`<br>`With sentences.`<br>`In it.`<br>`Additional line.` |
| **writelines()** | Write list of strings to a file. |  `string_list = ['\nYet another line.',`<br>`               '\nSo many lines',`<br>`               '\nVery many lines.']`<br><br>`with open('file.txt', 'a') as f:`<br>`    f.writelines(string_list)`<br><br>`with open('file.txt', 'r') as f:`<br>`    print(f.read())`<br><br>`Sample file.`<br>`With sentences.`<br>`In it.`<br>`Additional line.`<br>`Yet another line.`<br>`So many lines`<br>`Very many lines.` |