# THEORY QUESTIONS ASSIGNMENT

Python based theory

To be completed at student's own pace and submitted before given deadline

| NO | TASK | POINTS |
|---|---|---|
| **PYTHON** | | |
| 1 | Theory questions | 30 |
| 2 | String methods | 29 |
| 3 | List methods | 11 |
| 4 | Dictionary methods | 11 |
| 5 | Tuple methods | 2 |
| 6 | Set methods | 12 |
| 7 | File methods | 5 |
| | **TOTAL** | **100** |

| **1. Python theory questions** | **30 points** |
|---|---|

**1. What is Python and what are its main features?**
Python is a high-level multi-paradigm (not only Object Oriented) programming language. One of the easiest to learn and one of the highest number of libraries to work with. Libraries help a lot in saving time as the code can be reused and there's no need to write everything from scratch.

**2. Discuss the difference between Python 2 and Python 3**
Python 3 is more in-demand today and has simpler syntax. Python 2 is outdated and uses an older syntax. Maind syntax differences: Division, print operator, Unicode P3 instead of ASCII P2, xrange, error handling, __future__ module

**3. What is PEP 8?**
Python Enterprise Proposal -8 is a document that provides guidelines and best practices on how to write Python code

**4. In computing / computer science what is a program?**
Program is a step by step instruction that tells the computer what to do.

**5. In computing / computer science what is a process?**
A process becomes a program when it is loaded to the computer memory and is

ready to run. Program will be passive (just the code written) and process is active(the actual execution)

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

Cashe is a small amount of memory which is a part of the CPU and is probably will be used again at some point

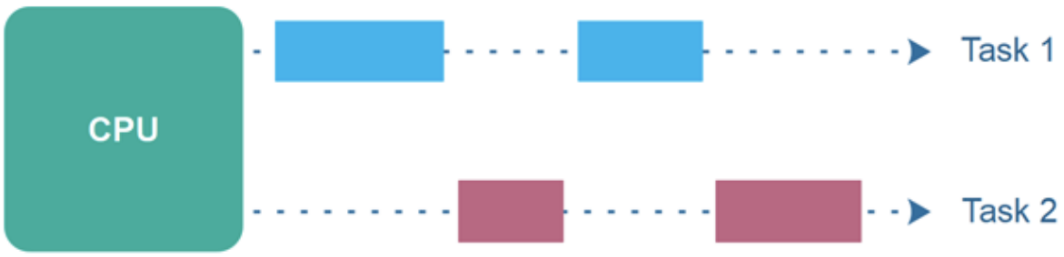
**7. In computing / computer science what is a thread and what do we mean by multithreading?**

A thread is a short executable part of a larger code. When multiple threads are run in a process at the same time, we get "multithreading."

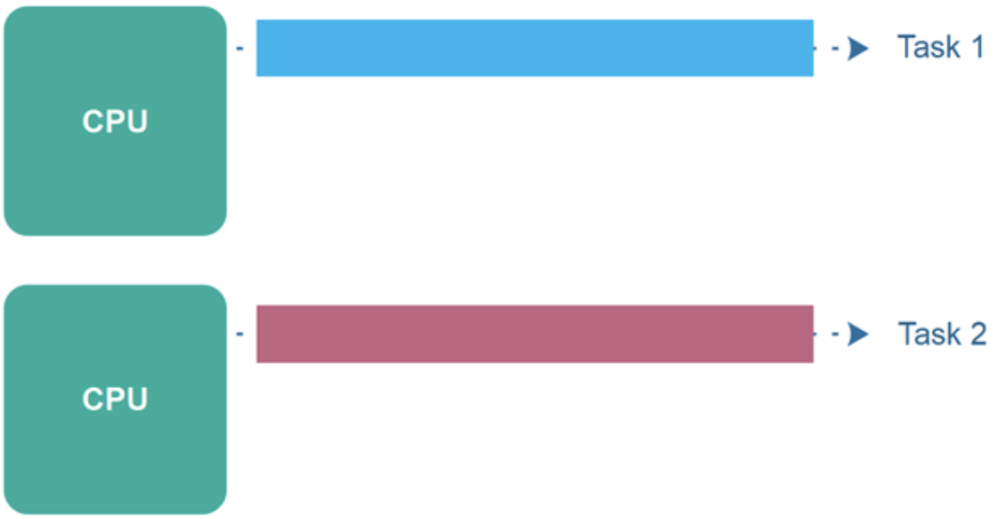
**8. In computing / computer science what is concurrency and parallelism and what are the differences?**

Concurrency means multiple computations are happening at the same time. Concurrency is the task of running and managing the multiple computations at the same time. While parallelism is the task of running multiple computations simultaneously. Visuals describe it the best:

Concurrent execution




Parallel Execution




(source for visuals: http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html)

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

Global Interpreter Lock is the lock that protects the interpreter which is not

thread-safe. It ensures that only one thread is running at any given time.

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

DRY - Don't Repeat Yourself. In practice this principle requires to avoid dublications when possible and use functions to wrap up the code.
KISS - Keep it Simple, Stupid. In practice - simplify things where possible.
BDUF - Big Design Up Front. In practice not a very cool thing to do, as it is about thinking too much, before actually doing anything.

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

GC manages the allocation and release of memory for a program. Python has an automated GC, it releases the memory automatically when the object is not in use.

**12. How is memory managed in Python?**

In Python the memory is managed by the memory manager. It manages the memory allocation based on its object-specific allocators, allocating memory distinctly for specific objects(int, string etc) and it manages the Python heap ( contains all Python objects and data structures) on demand. The Python memory manager manages chunks of memory called "Blocks". A collection of blocks of the same size makes up the "Pool". Pools are created on Arenas, chunks of 256kB memory allocated on heap=64 pools. Python also has different memory types: Methods and variables are created in *Stack memory*. Objects and instance variables are created in *Heap memory*. When memory is released the memory manager doesn't release it back to the python interpreter, and not the OS.

**13. What is a Python module?**

Module is a file containing a set of functions we want to include in our code. It can be user-defined or pre-built stored in libraries.

**14. What is docstring in Python?**

Docstrings are the string literals that appear after the definition of a function, method, class, or module. We use them to write documentation for our programs, describing the functionality. It can come straight after the function def and is inserted using 3 pairs of double quotes. In can be accessed by help() function or by calling __doc__.

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

Pickling is a way to convert a python object into a character stream so that this character stream contains all the information necessary to reconstruct the object in another python script.
Pickling: a process where a Python object hierarchy is converted into a byte stream.
Unpickling: a process where a byte stream is converted into an object hierarchy.
It is used to easily transfer data from one server/system to another and then store it in a file or database.

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

Pylint is a tool for static code analysis and bug detection. It looks for programming errors and is used for coding standards. It is highly configurable and acts like special programs to control warnings and errors. It can be integrated in various python IDEs.
Pychecker is an open source tool for static analysis that detects the bugs from source code and sends warnings about the style and complexity of the bugs.

**17. How are arguments passed in Python by value or by reference?** Give an example.
Python uses a different approach which is usually referred to as "Pass by Object Reference". The functions in python receive the reference of the same object in the memory as referred by the caller. Similar to pass-by-value, the function provides its own bucket and creates an entirely new variable for itself. Any changes made to the function variable(bucket) won't change the caller variable (bucket), only the content gets updated.

**18. What are Dictionary and List comprehensions in Python? Provide examples.**
Comprehension is writing new lists and dictionaries based on the values of the existing ones in a more concise way using simpler syntax.
E.g for lists:

```
programs = ["Data Science", "Software Engineering", "Data Analysis", "Data Engineering", "ML Engineering"]
engineering_programs = [x for x in programs if "Engineering" in x]
print(engineering_programs)
```

Instead of using the for statement like this:

```
programs = ["Data Science", "Software Engineering", "Data Analysis", "Data Engineering", "ML Engineering"]
engineering_programs = []
for x in programs:
  if "Engineering" in x:
    engineering_programs.append(x)
print(engineering_programs)
```

We can also create a comprehension for the dictionary from lists:
In our example we will create a dictionary to show how many students is enrolled for each engineering program- keys is our engineering programs list and values is the number of students

```
keys = ["Software Engineering","Data Engineering", "ML Engineering"]
values = [23, 12, 8]
engineering_enrollments = {k:v for (k,v) in zip(keys, values)}
print(engineering_enrollments)
```

**19. What is namespace in Python?**

Namespace is a system that has a unique name for each and every object in Python. There are three types of Python namespaces- global, local, and built-in.Name spaces help to prevent conficts in Python.

**20.What is pass in Python?**

Pass is used as a placeholder for a future code. Empty code is not allowed in loops, function definitions, class definitions, or in if statements. So if we need to run the empty code with them without an error, we can use pass, nothing will happen but the rest of the code can run smoothly.

**21. What is unit test in Python?**

Unit testing is a method used in software testing when individual units of source code are put under various tests to check whether they are fit for use. Python uses framework unittest which comes as a part python package. We can test individual function, module, class or method.

**22. In Python what is slicing**?

Slicing is accessing a specific portion or a subset of the list/string for some operation while the initial list/string remains unaffected.

**23. What is a negative index in Python?**

Negative index is an indication of what position the object has if we start counting from the end.

E.g.

My_list = [chocolate, candy, cake]

The regular index for candy will be 1, the negative index will be -2.

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

Using ternary operators is a way of writing conditional statements in Python. These operators evaluate whether the condition is true or false in one line.

E.g

```
a, b = 15, 30
max = a if a > b else b
print(max)
```

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

We use *args and **kwargs as arguments of a function when we are unsure about the number of arguments to pass in the functions.

using *args we can pass a variable number of non-keyword arguments to a Python function.

Using **kwargs we can pass a variable number of keyword arguments to a Python function. We need to remember to use unpacking operators * for argos or ** for kwargs.

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

They both generate a list of integers, but the difference is that range returns a

Python list object and xrange returns an xrange object. xrange creates a non-static list adding the values as we need them with a special technique called yielding. xrange is good to use if we work with really large ranges.

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

Flask is a third-party Python library used for developing web applications.

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

Both clustered and non-clustered indexes contain only keys and record identifiers in the index structure. However, cluster index is a type of index which sorts the data rows in the table based on their key values. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.

**29. What is a 'deadlock' a relational database?**

When two or more transactions are waiting for one another to give up locks.

**30.What is a 'livelock' a relational database?**

When two or more transactions change their state continuously, with neither of them making progress

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

Methods theory source: w3schools, examples own

| METHOD | DESCRIPTION | EXAMPLE |
|---|---|---|
| capitalize() | Converts the first character to upper case | bookshop_welcome = "welcome to my bookshop Coding Fairytales."<br>cap_welcome = bookshop_welcome.capitalize()<br>print (cap_welcome) |
| casefold() | Converts string into lower case<br>This method differs from lower() as it can also convert special characters into lowercase. | bookshop_welcome = "Welcome To My Bookshop Coding Fairytales."<br>cf_welcome = bookshop_welcome.casefold()<br>print (cf_welcome) |
| center() | Returns a centered string | bookshop_name = "Coding Fairytales"<br><br>put_to_center = bookshop_name.center(20)<br><br>print(put_to_center) |

| | | |
|---|---|---|
| count() | Returns the number of times a specified value occurs in a string | bookshop_name = "Coding Fairytales"<br><br>i_count = bookshop_name.count(i)<br><br>print(i_count) |
| endswith() | Returns true if the string ends with the specified value | bookshop_name = "Coding Fairytales"<br><br>endswith_fs = bookshop_name.endswith(“Fairytales”)<br><br>print(endswith_fs) #True |
| find() | Searches the string for a specified value and returns the position of where it was found | bookshop_name = "Coding Fairytales"<br><br>where_is_f = bookshop_name.find(“F”)<br><br>print(where_is_f) |
| format() | Formats specified values in a string | book_price_msg = "For only {price:.2f} dollars!"<br>print(book_price_msg.format(price = 49)) |
| index() | Searches the string for a specified value and returns the position of where it was found | bookshop_name = "Coding Fairytales"<br><br>index_f = bookshop_name.index(“F”)<br><br>print(index_f) |
| isalnum() | Returns True if all characters in the string are alphanumeric | book_name = "Coding247"<br><br>x= book_name.isalnum()<br><br>print(x) #True |

| | | |
|---|---|---|
| isalpha() | Returns True if all characters in the string are in the alphabet | book_name = "Coding247"<br><br>x= book_name.isalpha()<br><br>print(x) #False |
| isdigit() | Returns True if all characters in the string are digits | book_name = "Coding247"<br><br>x= book_name.isdigit()<br><br>print(x) #False<br>book_name_2 = "3,14" |

| | | |
|---|---|---|
| | | x= book_name_2.isdigit()<br><br>print(x) #True |
| islower() | Returns True if all characters in the string are lower case | bookshop_name = "Coding Fairytales"<br><br>lower = bookshop_name.lower()<br><br>print(lower) |
| isnumeric() | Returns True if all characters in the string are numeric | stock = 667<br>store_in_num = stock.isnumeric()<br>print(store_in_num) #True |
| isspace() | Returns True if all characters in the string are whitespaces | sample_text = " _ "<br>x = sample_text.isspace()<br>print(x) |
| istitle() | Returns True if the string follows the rules of a title | bookshop_name = "Coding Fairytales"<br><br>is_title = bookshop_name.istitle()<br><br>print(is_title) #True |
| isupper() | Returns True if all characters in the string are upper case | bookshop_name = "Coding Fairytales"<br><br>loud_title = bookshop_name.isupper()<br><br>print(loud_title) #False |
| join() | Converts the elements of an iterable into a string | Books_Collection = ("Python Adventures", "How SQL Lost the Keys ", "How  Everything Worked with no Errors")<br><br>i_will_read_about = " &<br>".join(Books_Collection)<br><br>print(i_will_read_about) |
| lower() | Converts a string into lower case | bookshop_name = "Coding Fairytales"<br><br>quiet_title = bookshop_name.lower()<br><br>print(quiet_title) |
| lstrip() | Returns a left trim version of the string | fav = "  Python Adventures  "<br><br>fav_book = fav.lstrip() |

| | | |
|---|---|---|
| | | print("of all books", fav_book, "is my favorite") |
| replace() | Returns a string where a specified value is replaced with a specified value | book_title = "How SQL lost all the keys"<br><br>right_book_title = book_title.replace("lost", "found")<br><br>print(right_book_title) |
| rsplit() | Splits the string at the specified separator, and returns a list | book_collection = " Python Adventures, How SQL Lost all the Keys, How  Everything Worked with no Errors. "<br><br>pretty_book_collection = book_collection.rsplit(", ")<br>print(pretty_book_collection) |
| rstrip() | Returns a right trim version of the string | fav = "  Python Adventures  "<br><br>fav_book = fav.rstrip()<br>print("of all books", fav_book, "is my favorite") |
| split() | Splits the string at the specified separator, and returns a list | bookshop_welcome = "Welcome to my bookshop Coding Fairytales."<br>slow_welcome =bookshop_welcome.split()<br>print (slow_welcome) |
| splitlines() | Splits the string at line breaks and returns a list | bookshop_welcome = "Welcome to my bookshop Coding Fairytales.\n Our books are amazing"<br>separate_welcome =bookshop_welcome.splitlines()<br>print (separate_welcome) |
| startswith() | Returns true if the string starts with the specified value | bookshop_welcome = "Welcome to my bookshop Coding Fairytales.\n Our books are amazing"<br>starts_welcome = bookshop_welcome.startswith("Welcome")<br>print (starts_welcome)<br>#True |
| strip() | Returns a trimmed version of the string | fav = "  Python Adventures  " |

| | | |
|---|---|---|
| | | fav_book = fav.rstrip()<br>print("of all books", fav_book, "is my favorite") |
| swapcase() | Swaps cases, lower case becomes upper case and vice versa | bookshop_welcome = "Welcome to my bookshop Coding Fairytales."<br>fun_welcome = bookshop_welcome.swapcase()<br>print (fun_welcome) |

| | | |
|---|---|---|
| title() | Converts the first character of each word to uppercase | Books_Collection = "python adventures, how sQL lost the keys, how everything worked with no errors"<br><br>proper_titles = Books_Collection.title()<br><br>print(proper_titles) |
| upper() | Converts a string into upper case | bookshop_name = "coding fairytales"<br><br>loud_title = bookshop_name.upper()<br><br>print(loud_title) |

| | |
|---|---|
| 3. Python list methods:<br>describe each method and provide an example | 11 points |

| **Method** | **Description** | **Example** |
|---|---|---|
| append() | Converts a string into upper case | book_collection = ["Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"]<br>book_collection.append(" Fantastic Lists and How to Handle Them")<br>print(book_collection) |
| clear() | Removes all the elements from the list | book_collection = ["Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"]<br>book_collection.clear() |

| | | |
|---|---|---|
| | | sold_out = ["sold out"]<br>print(book_collection + sold_out) |
| copy() | Returns a copy of the list | book_collection = ["Python Adventures", "How SQL Lost all the Keys", "How  Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"]<br>book_collection.copy()<br>print("now we have two of each")<br>print(book_collection) |
| count() | Returns the number of elements with the specified value | book_collection = ["Python Adventures", "How SQL Lost all the Keys", "How  Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"]<br>python_shelf = book_collection.count("Python Adventures")<br><br>print(python_shelf) |
| extend() | Add the elements of a list (or any iterable), to the end of the current list | book_collection = ["Python Adventures", "How SQL Found all the Keys", "How  Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"]<br>new_arrival = ["Pandas lost in the library", "Pip the almighty"]<br>book_collection.extend(new_arrival)<br>print(book_collection) |
| index() | Returns the index of the first element with the specified value | book_collection = ["Python Adventures", "How SQL Found all the Keys", "How  Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"]<br>lists_shelf = book_collection.index("Fantastic Lists and How to Handle Them")<br>print(lists_shelf)<br>#3 |
| insert() | Adds an element at the specified position | book_collection = ["Python Adventures", "How SQL Found all the Keys", "How  Everything Worked with no Errors"]<br>book_collection.insert(1, "Fantastic Lists and How to Handle Them")<br><br>print(book_collection) |
| pop() | Removes the element at the specified position | book_collection = ["Python Adventures", "Fantastic Lists and How to Handle Them", |

| | | |
|---|---|---|
| | | "How SQL Found all the Keys", "How Everything Worked with no Errors"]<br>book_collection.pop(1)<br><br>print(book_collection) |
| remove() | Removes the item with the specified value | book_collection = ["Python Adventures", "Fantastic Lists and How to Handle Them", "How SQL Found all the Keys", "How Everything Worked with no Errors"]<br>book_collection.remove("Python Adventures")<br><br>print(book_collection) |
| reverse() | Reverses the order of the list | book_collection = ["Python Adventures", "Fantastic Lists and How to Handle Them", "How SQL Found all the Keys", "How Everything Worked with no Errors"]<br>book_collection.reverse()<br><br>print(book_collection) |
| sort() | Sorts the list | book_collection = ["Python Adventures", "Fantastic Lists and How to Handle Them", "How SQL Found all the Keys", "How Everything Worked with no Errors"]<br>book_collection.sort()<br><br>print(book_collection) #sorted alphabethically |

| | |
|---|---|
| 4. Python tuple methods:<br>describe each method and provide an example | 2 points |

| **Method** | **Description** | **Example** |
|---|---|---|
| count() | Returns the number of times a specified value occurs in a tuple | book_collection = ("Python Adventures", "Fantastic Lists and How to Handle Them", "How SQL Found all the Keys", "How  Everything Worked with no Errors")<br>how_count = book_collection.count("How") |

| | | |
|---|---|---|
| | | print(how_count)<br>#0, because immutable |
| index() | Searches the tuple for a specified value and returns the position of where it was found | book_collection = ( "Fantastic Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys", "How  Everything Worked with no Errors")<br>python_shelf = book_collection.index("Python Adventures")<br><br>print(python_shelf) |

| | |
|---|---|
| 5. Python dictionary methods:<br>describe each method and provide an example | 11 points |

| Method | Description | Example |
|---|---|---|
| clear() | Removes all the elements from the dictionary | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>books_inventory.clear()<br>print(books_inventory)<br>#empty dict |
| copy() | Returns a copy of the dictionary | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>books_inventory.copy()<br>print(books_inventory) |
| fromkeys() | Returns a dictionary with the specified keys and value | books_collection = ("Python Adventures" , "How SQL Found all the Keys", "How  Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pandas lost in the library", "Pip the almighty")<br>actions = ("buy", "borrow", "gift") |

| | | |
|---|---|---|
| | | books_actions = dict.fromkeys(books_collection,actions)<br>print(books_actions) |
| get() | Returns the value of the specified key | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>python_numbers = books_inventory.get("Python Adventures")<br>print(python_numbers) |
| items() | Returns a list containing a tuple for each key value pair | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>tuppled = books_inventory.items()<br>print(tuppled) |
| keys() | Returns a list containing the dictionary's keys | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>names_list = books_inventory.keys()<br>print(names_list) |
| pop() | Removes the element with the specified key | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>sold_book = books_inventory.pop("Fantastic Lists and How to Handle Them")<br>print(books_inventory) |
| popitem() | Removes the last inserted key-value pair | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>books_inventory.popitem()<br>print(books_inventory) |

| | | |
|---|---|---|
| | | |

| | | |
|---|---|---|
| setdefault() | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>pip_left = books_inventory.setdefault("Pip the almighty")<br>print(pip_left) |
| update() | Updates the dictionary with the specified key-value pairs | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>pip_left = books_inventory.update({"Magic with functions": "10"})<br>print(books_inventory) |
| values() | Returns a list of all the values in the dictionary | books_inventory = {"Python Adventures": 4 , "How SQL Found all the Keys": 8, "How  Everything Worked with no Errors":1, "Fantastic Lists and How to Handle Them": 0, "Pandas lost in the library": 3, "Pip the almighty": 2}<br>book_numbers = books_inventory.values()<br>print(book_numbers) |

| | |
|---|---|
| 6. Python set methods:<br>describe each method and provide an example | 12 points |

| Method | Description | Example |
|---|---|---|
| add() | Adds an element to the set | book_collection = {"Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"}<br>book_collection.add(" Fantastic Lists and How to Handle Them")<br>print(book_collection) |

| | | |
|---|---|---|
| clear() | Removes all the elements from the set | book_collection = {"Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"}<br>book_collection.clear()<br>print(book_collection) |
| copy() | Returns a copy of the set | book_collection = {"Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"}<br>book_collection.copy()<br>print(book_collection) |
| difference() | Returns a set containing the difference between two or more sets | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"}<br>out_of_stock = {"Pandas lost in the library", "Pip the almighty", "Python Adventures"}<br><br>current_collection = book_collection.difference(out_of_stock)<br><br>print(current_collection) |
| intersection() | Returns a set, that is the intersection of two or more sets | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them"}<br>book_collection_newbranch = {"Fantastic Lists and How to Handle Them", "Pandas lost in the library", "Pip the almighty", "Python Adventures"}<br><br>available_in_both = book_collection.intersection(book_collection_newbranch)<br><br>print(available_in_both) |
| issubset() | Returns whether another set contains this set or not | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pip the almighty", "Pandas lost in the library"}<br>book_collection_newbranch = {"Fantastic |

| | | |
|---|---|---|
| | | Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys"}<br><br>storage_in_main_collection = book_collection_newbranch.issubset(book_collection)<br><br>print(storage_in_main_collection) |
| issuperset() | Returns whether this set contains another set or not | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pip the almighty", "Pandas lost in the library"}<br>book_collection_newbranch = {"Fantastic Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys"}<br><br>storage_in_main_collection = book_collection.issuperset(book_collection_newbranch)<br><br>print(storage_in_main_collection) |
| pop() | Removes an element from the set | book_collection = {"Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"}<br>book_collection.pop()<br>print(book_collection) #randomly deleted item/every time different one |
| remove() | Removes the specified element | book_collection = {"Python Adventures", "How SQL Lost all the Keys", "How Everything Worked with no Errors"}<br>book_collection.remove("Python Adventures")<br>print(book_collection) |
| symmetric_differe r ence() | Returns a set with the symmetric differences of two sets | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pip the almighty", "Pandas lost in the library"}<br>book_collection_newbranch = {"Fantastic |

| | | |
|---|---|---|
| | | Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys"}<br><br>different_collections = book_collection.symmetric_difference(book_collection_newbranch)<br><br>print(different_collections) |
| union() | Return a set containing the union of sets | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pip the almighty", "Pandas lost in the library"}<br>book_collection_newbranch = {"Fantastic Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys"}<br><br>all_books_in_storage = book_collection.union(book_collection_newbranch)<br><br>print(all_books_in_storage) |

| | | |
|---|---|---|
| update() | Update the set with another set, or any other iterable | book_collection = {"Python Adventures", "How SQL Found all the Keys", "How Everything Worked with no Errors", "Fantastic Lists and How to Handle Them", "Pip the almighty", "Pandas lost in the library"}<br>book_collection_newbranch = {"Fantastic Lists and How to Handle Them", "Python Adventures", "How SQL Found all the Keys"}<br><br>book_collection.update(book_collection_newbranch)<br><br>print(book_collection) |

| 7. Python file methods:<br>describe each method and provide an example | 5 points |
|---|---|

| Method | Description | Example |
|---|---|---|
| read() | Returns the file content | f = open("Fantastic Lists and How to Handle Them.txt", "r")<br>print(f.read()) |
| readline() | Returns one line from the file | f = open("Fantastic Lists and How to Handle Them.txt", "r")<br>print(f.readlines()) |
| readlines() | Returns a list of lines from the file | f = open("Fantastic Lists and How to Handle Them.txt", "r")<br>print(f.readline()) |
| write() | Writes the specified string to the file | f = open("Fantastic Lists and How to Handle Them.txt", "a")<br>f.write("And then....they got concatenated!")<br>f.close() |
| writelines() | Writes a list of strings to the file | f = open("Fantastic Lists and How to Handle Them", "a")<br>f.writelines(["And then....they got concateneted!", "Over and over again...."])<br>f.close() |