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 an interpreted, interactive, object-oriented, open-source programming language used to build applications and handle data. It is beginner-friendly, free, intuitive, and robust language, making it very attractive for professionals to prototype using it. It is one of the most popular programming languages nowadays, and it is used in different areas: Data science, data analysis, machine learning, wes development etc.

Main features:

- High-level programming language, with a simple syntax, which makes it easier to learn;
- Free and open-source;
- Object-oriented language.
- A large supportive community, constantly contributing with the libraries and modules.

- Interpreted language, there is no need for compilation, and it is easier to debug.

2. Discuss the difference between Python 2 and Python 3

Python 3 presents many changes. For instance, it has an easier syntax. In Python 2, a division between integers returns another integer, and in Python 3, the same procedure will return the actual value as a float.
Another difference applies to the print keyword statement in Python 2, evolving to the print() function. In Python 3, strings are stored as UNICODE by default.

3. What is PEP 8?

PEP 8 stands for Python Enhancement Proposal 8. It is a style guide to writing clean and readable code based on the best practices described in the convention's guidelines.

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

It is a precise set of instructions that tells the computer exactly what to do step-by-step, to complete a specific task.

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

A process Is an instance of an executing program created when a program is executed to perform tasks mentioned in the program, is loaded into the main memory, and exists for a limited period, terminating after completing the job.

6. In computing / computer science what is cache?

Cache is a small portion of memory used to temporarily store instructions and data that is likely to be reused.

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

It's a set of simultaneous tasks contained inside a process. A process can have multiple threads, which is called multithreading.

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

Concurrency means multiple tasks that start, run, and complete in overlying periods, in no exact order. Parallelism is when various tasks or several parts of a unique task run at the same time. Concurrency alternates between different tasks and deals with many things at a time. Parallelism divides a task into smaller sub-tasks and processes it simultaneously or in parallel.

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

GIL stands for Global Interpreter Lock. A GIL creates a lock that ensure there is only one thread executing, to avoid sharing code that is not thread-safe with other threads.

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

DRY: Don't repeat yourself. The goal it's to avoid repetition of information.
KISS: Keep it super simple. Aim to keep everything as simple as possible and efficient.
BDUF: Big Design Up Front encourages to create a detailed design of the project before coding

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

Python Garbage Collector automatically manages an application's allocation and release of memory.
Garbage collection is implemented in Python in two forms: reference counting and generational. When the reference count of an object reaches 0, the reference counting garbage collection algorithm cleans up the object instantly. The generational garbage collection algorithm runs and cleans the object if a cycle reference count doesn't reach zero.

12. How is memory managed in Python?

Python uses the dynamic memory allocation, which is managed by the heap data structure. Memory Heap keeps the objects and other data structures that will be used in the program. The garbage collector automatically handles Python memory allocation and deallocation procedures.

13. What is a Python module?

A Python module is a file containing a set of functions definitions, classes, variables, Python statements, and can also include executable code.

14. What is docstring in Python?

A Python docstring is a string used to document a Python module, class, function or method. Docstrings are declared using" triple single quotes or triple double quotes, and should de created right after the class, method or function declaration.
Docstrings can be accessed using the __doc__ method of the object.

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

Pickling is the process of converting any type of python object: sets, dict, lists, tuple etc., into a byte stream of 0s and 1s. We can convert the byte stream (generated via pickling) back into python objects by the inverse process named unpickling. Pickling and unpickling are alternatively known as "serialization", "marshalling," or "flattening. Can be used for example to storing Python objects in a database.

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

You can use the IDLE Debug control window or additional tools to perform debugging and statistic analysis, such as PyChecker and Pylint.

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

All arguments in the Python language are "Pass by Object Reference". It means that if you change what a parameter refers to within a function, the change reflects back in the calling function.

Example:

```
fruit_basket={'Apple':1 ,'Mango':1.5, Grapes':2}
def test(fruit_basket):
  new={'Papaya':3}
  fruit_basket.update(new)
  print("Example",fruit_basket)
  return
test(fruit_basket)
print("Example 2:",fruit_basket)
```

18. What are Dictionary and List comprehensions in Python? Provide examples.

Python Dictionary and Lists comprehensions are an agile way of creating a list or dictionary in Python. It allows us to loop through existing data and filter it to create a list or dictionary. In short, it's a way to make things clear and concise to express.
Example:

```
values = [2, 4, 6, 8, 10]
doubled_values = [x*2 for x in values]
print(doubled_values)
# Outputs [4, 8, 12, 16, 20]
```

19. What is namespace in Python?

A namespace is a compilation of names and the attributes of the objects referenced by the names. We can consider a namespace as a Python dictionary that maps object names to objects. The dictionary's keys correspond to the names, and the values correspond to the objects.
There are four types of namespaces: built-in namespaces, global namespaces, local namespaces and enclosing namespaces.

20.What is pass in Python?

In Python, a pass is a null operation and allows execution to continue. The interpreter does not ignore a pass statement, but nothing happens, and the statement results in no process.

21. What is unit test in Python?

The purpose of a unit test is to verify the behaviour of a relatively small piece of software, independently from other parts to ensures that it performs as expected.

22. In Python what is slicing?

Slice is a function that returns a slice object that can be used to slice any sequence (string, tuple, list, etc.). It is possible to set where to start the slicing and where to end. It is also possible to determine the step which allows slicing only every other item

23. What is a negative index in Python?

Negative index is a way to pick the data from the end of a list and can also be used to invert a number or string. This means the latest value of a sequence holds an index of -1, the second last -2, etc.

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

The ternary operator is a way of writing conditional statements in Python.

Example:

```
If age >= 12:
print("You are a teenager")
else:
("You are not a teenager")
```

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

*args and **kwargs are special symbols used for passing arguments to a function, and can be used when we have doubts about the number of arguments we should pass in a function.

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

The range() and xrange() are functions that can be used to iterate a certain number of times in for loops in Python.
() range uses more memory as it keeps the entire list of elements in memory.
()xrange uses less memory as it only keeps one element at a time in memory.

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

Flask is a backend popular micro lightweight framework written in Python and is used for building web applications.

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

A Clustered index is a kind of index where table records are physically reordered to match the index. A Non-Clustered index is a particular type of index where the logical order of the index does not match the stored physical order of the rows on the disk.

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

A deadlock is a blocking scenario when two concurrent transactions compete for exclusive access to the same resource and cannot obtain it.

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

A Livelock is a situation where a request for an exclusive lock is denied repeatedly, as many overlapping shared locks keep on interfering each other.

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

capitalize()		
casefold()		
center()		
count()		
endswith()		
find()		
format()		
index()		
isalnum()		

isalpha()		
isdigit()		
islower()		
isnumeric()		
isspace()		
istitle()		
isupper()		
join()		

lower()		
lstrip()		
replace()		
rsplit()		
rstrip()		
split()		
splitlines()		
startswith()		
strip()		
swapcase()		

title()		
upper()		

Python string methods:

`capitalize()`

Returns a string where the first character is upper case, and the rest is lower case.

Example:

```
my_example  =  "hello, stranger.""
example = my_example.capitalize()
print(example)

# prints: Hello, stranger.
```

casefold()

Returns a string where all the characters are lower case.

Example:

```
my_example  =  " HELLO, Stranger.""
example = my_example.casefold()
print(example)

# prints: hello, stranger.
```

center()

Returns a new centered string after padding it with the specified character (space is default).

Example:

```
my_string  =  " Hello, Vania."
new_string = my_string.center(30, '#')
print(new_example)

# prints:  ######## Hello, Vania.########
```

count()

Example:

Returns the number of times a specified value appears in the string.

```
my_string  =  "  Summer is my favourite season. I love the summer.""
example = my_string.count("summer")
print(example)

# prints: 1
```

endswith()

Returns True if a string ends with the specified suffix. If not, it returns False.

Example:

```
message = 'Today is Sunday.'
print(message.endswith('Sunday'))

# prints: True
```

find()

Returns the index of first occurrence of the substring (if found). If not found, returns -1.

Example:

```
message = 'Today is a good day to learn Python.'

print(message.find('day'))

# prints: 2
```

format()

Returns the formatted string.

It is a method that formats the specified value(s) and inserts them inside the string's placeholder. The placeholder is defined using curly brackets: {}

Example:

```
name = 'Vania'
age = '37'
print("My name is {name}. I'm {age} years old. ".format(name = "Vania", age = 37))

# prints: My name is Vania. I'm 37 years old.
```

index()

Returns the index of a substring inside the string (if found) , if the substring is not found, it will raise an exception.

Example:

```
example = 'My favourite colour is blue.'
result = example.index('is')
print(result)
```

```
# prints: 20
```

isalnum()

Returns True if all the characters are alphanumeric: alphabet letters (a-z) and numbers (0-9), if not, will return False.

Example:

```
example = "Year2022"
print(example.isalnum())

# prints: True
```
(If it were Year 2022, would print False given the space)

isalpha()

Returns True if all characters in the string are alphabets. If not, it returns False.

Example:
```
name = "Vania3"
print(name.isalpha()

# prints: False
```

isdigit()

Returns True if all the characters in the string are digits; otherwise, returns False.

Example:

```
example = 'python2'
print(example.isdigit())

# prints: False
```

islower()

Returns True if all alphabets in a string are lowercase, if contains one uppercase alphabet, it will return False.

Example:

```
hi = 'Hi, how are you?'
print(hi.islower())

# prints: False
```

isnumeric()

Returns True if all characters in a string are numeric characters. If not, it returns False.

Example:

```
example = 'Vania123'
print(example.isnumeric())

# Prints: False
```

isspace()

Returns True if there are only whitespace characters in the string. If not, it returns False.

Example:

```
example = ' hi '
print(example.isspace())

# prints: False
```

istitle()

Returns True if the string is title-cased. If not, it returns False.

Example:

```
example = 'I am Vania.'
print(example.istitle())

# prints: False
```

isupper()

Returns True if all the characters in a string are in upper case. Otherwise, returns False.

Example:

```
example = "IS This uppercase?!"
print(example.isupper())

# prints: False
```

join()

Takes all items in an iterable and joins them into one string.

Example:

```
example = ("Celo", "Vania", "Luna", "Mel")
my_example = "***".join(example)
print(my_example)

# prints: Celo***Vania***Luna***Mel
```

lower()

Returns a new string, entirely lowercase.

Example:

```
message = 'What DAY is today?'
print(message.lower())

# prints: what day is today?
```

lstrip()

Returns a copy of the string in which all chars have been stripped from the beginning and the end of the string ((based on the string argument passed).

Example:

```
example = '     My example.  '
print('Example:', example.strip())

# prints: Example: My example.
```

replace()

Returns a copy of the string where the old substring is replaced with a new one.

Example:

```
example = 'This is an example'
replaced_example = example.replace('is', 'was', 1)
print(replaced_example)

# prints: Thwas is an example
```

rsplit()

Returns a list of the words in the string, separated by the delimiter string (starting from the right).

Example:

```
example = 'Today is Sunday'
print(example.rsplit())

# prints: ['Today', 'is', 'Sunday']
```

rstrip()

Returns a copy of the string in which all characters have been stripped from the end of the string (default whitespace characters).

Example:

```
string = 'this is weird    '
print(string.rstrip(' eird '))

# prints: this is w
```

split()

This method splits the characters in a string into separate items in a list, breaking the given string by the specified separator.

Example:

```
example = 'Today is a sunny Sunday'
print(example.split(' '))


#prints:  ['Today', 'is', 'a', 'sunny', 'Sunday']
```

splitlines()

It is a method that splits the string at line breaks and returns a list of lines in the string. If there are no line break characters, it returns a list with a single item, in a single line.

Example:

```
fruits = 'Banana, Mango, Coconut, Grapes'

print(fruits.splitlines())

# prints: ['Banana, Mango, Coconut, Grapes']
```

startswith()

Returns True if a string starts with the specified prefix(string). Otherwise, it returns False.

Example:

```
example = 'My favourite colour is blue'
print(example.startswith('My'))

# prints: True
```

strip()

It is a method that removes any spaces or defined characters at the start and end of a string and returns a new string without the characters specified.

Example:

```
my_example = "***This is my example.***"
print(my_example.strip("*"))

# prints: This is my example.
```

swapcase()

Converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string and returns it.

Example:

```
my_example = "CGF SUMMER degree."
print(my_example.swapcase())
```

```
# prints: cgf summer DEGREE.
```

title()

Returns a string where the first character in every word is upper case, like a header or a title.

Example:

```
my_example = 'My favourite colour is blue.'
print(my_example.title())

# prints: My Favourite Colour Is Blue.
```

upper()

Returns a string where all characters are in upper case. Symbols and numbers are ignored.

Example:

```
my_example = "cfg summer degree 2022!"
print(my_example.upper())

# prints: CFG SUMMER DEGREE 2022!
```

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

append()		
clear()		
copy()		
count()		
extend()		
index()		
insert()		
pop()		
remove()		
reverse()		
sort()		

append()

It is a method that you can use to add items to the end of a given list.

Example:

```
fruits = ['Apple', 'Grapes', 'Melon']
fruits.append('Papaya')
print(fruits)

# prints: ['Apple', 'Grapes', 'Melon', 'Papaya']
```

clear()

It's a method that removes all the elements from a list.

Example:

```
numbers = [1, 2, 3, 4, 5, 6, 7]
numbers.clear()
print(numbers)

# prints:  [ ]
```

copy()

This method returns a new list. It doesn't modify the original list, and does not accept any parameters.

Example:

```
my_example = ['Pen', 'notebook', 21, 6]
my_example = my_example.copy()
print('Copy of the list:', my_example)

# prints:  Copy of the list: ['Pen', 'notebook', 21, 6]
```

count()

Returns the number of times the specified element appears in the list.

Example:

```
bakery = ['Pie', 'Chocolat', 'Biscuits', 'Cupcake', 'Cake', 'Pie']
count = bakery.count('Pie')
print( count)

# prints:  2
```

extend()

Extend() is a method that adds all the elements of an iterable (tuple, string, list etc.) to the end of the list.
Example:

```
week_days = ['Mon', 'Tue']
week_days_extended = ['Wed', 'Thur', 'Fri', 'Sat']
week_days.extend(week_days_extended)
print(week_days)
```

```
# prints:  (List extended: ['Mon', 'Tue', 'Wed', 'Thur', 'Fri', 'Sat'])
```

index()

The index() is a method that returns the index of a specific element in a list (it only returns the first occurrence of the matching element). If the given item is not found, a ValueError exception is raised.

Example:

```
week_days = ['Mon', 'Tue', 'Wed', 'Thur', 'Fri', 'Mon']
index = week_days.index('Mon')
print('Index of Mon:', index)

# prints:  Index of Mon: 0
```

insert()

The insert() method takes two parameters. The first is the element's index; the second is the element to be inserted. It allows us to add a specific element at a specified index of the list.

Example:

```
fruits = ['Apple', 'Grapes', 'Mango']
fruits.insert(1, 'Papaya')
print('List:". fruits)

# prints:  List: ['Apple', 'Papaya', 'Grapes', 'Mango']
```

pop()

It is a method that removes the item at the specified position in the list and return it. If no index is specified, it will remove and returns the last item in the list.

Example:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8]
removed_number = numbers.pop(5)
print(removed_number)
print(numbers)

# prints: 6
[1, 2, 3, 4, 5, 7, 8]
```

remove()

It's a method that takes a single element as an argument and removes it from a list. It does not return any value, only removes the item from the list.

Example:

```
fruits = [ 'Apple', 'Mango', 'Kiwi', 'Papaya' ]
fruits.remove('Apple')
print(fruits)

# prints: ['Mango', 'Kiwi', 'Papaya']
```

reverse()

It is a method that reverses the elements of a list. It does not take any argument, and only updates the current list.

Example:

```
fruits = ['Apple', 'Mango', 'Kiwi']
fruits.reverse()
print('Fruits after reverse:', fruits)

# prints: Fruits after reverse: ['Kiwi', 'Mango', 'Apple']
```

sort()

Returns a sorted list of the iterable object in an ascending or descending order, depending on the specification. Strings are sorted alphabetically, and numbers are sorted numerically. It is not possible to sort a list containing simultaneous string and numeric values. By default, it doesn't require any additional parameters, but has two optional parameters:
reverse - If True, the sorted list is reversed (or sorted in Descending order)
key - function that serves as a key for the sort comparison

Example:

```
fruits = ['Apple', 'Mango', 'Kiwi', 'Banana']
fruits.sort()
print('Fruits after sort:', fruits)

# prints: Fruits after sort: ['Apple', 'Banana', 'Kiwi', 'Mango']
```

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

count()		
index()		

count()

The count() method returns the number of times the specified element appears in the tuple. It takes a single argument, which is the element to be counted.

Example:

```
numbers = (1, 3, 5, 7, 9, 1, 11, 1)
count = numbers.count(1)
print(count)

#prints: 3
```

index()

It is a method that returns the index of the first occurrence of a given element in a tuple; If not found, a ValueError exception is raised.

Example:

```
numbers = (1, 3, 5, 7, 9, 1, 11, 1, 7)
index = numbers.index(7)
print('The index of 7 is', index)

#prints: The index of 7 is 3
```

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

clear()		
copy()		
fromkeys()		
get()		
items()		
keys()		
pop()		
popitem()		

setdefault()		
update()		
values()		

clear()

This method removes all items from the dictionary.

Example:

```
my_example = {1985: "first", 2004: "second"}
my_example.clear()
print('my_example =', my_example)
# prints: my_example = {}
```

`copy()`

This is a method that returns a shallow copy of the *dictionary*.

Example:

```
my_example = {1985:'first', 2004:'second'}
new = my_example.copy()
print('My example: ', original)
print('New: ', new)

#prints: My example: {1985: 'first', 2004: 'second'}
New: {1985: 'first', 2004: 'second'}
```

`fromkeys()`

This method creates a new dictionary from the provided iterable (set, string, tuple) as keys and with the specified value.

Example:

```
keys = {'Apple', 'Mango', 'Banana', 'Orange', 'Grapes' }
value = 'fruit'
fruit = dict.fromkeys(keys, value)
print(fruit)

#prints: {'Orange': 'fruit', 'Banana': 'fruit', 'Apple': 'fruit', 'Mango': 'fruit', 'Grapes': 'fruit'}
```

`get()`

Returns the value for a key if it exists in the *dictionary*

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20}
print(students.get('Ana'))

#prints: 17
```

items()

This method returns a view object. The view object contains the key-value pairs of the dictionary as tuples in a list.

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20}
print(students.items())

#prints: dic_items([('Ana', 17), ('Mel', 18), ('Karl', 20)])
```

keys()

This method returns a view object. The view object contains the keys of the *dictionary* as a list.

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20}
print(students.keys())

#prints: dict_keys(['Ana', 'Mel', 'Karl'])
```

pop()

This method removes an element from the dictionary. It removes the element associated with the specified key.

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20}
students = students.pop('Karl')
print('Popped:', students)

#prints: Popped: 20
```

popitem()

This method removes and returns the last element (key, value) pair inserted into the dictionary.
Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20, 'Marco': 23}
students_pop = students.popitem()
print('Popped:', students

#prints: Popped: {'Marco"23}
```

setdefault()

This method returns the value of a key (if the key exists in the dictionary). If not, it will insert the key with a value to the dictionary.

update()

This method inserts specified items into the dictionary.

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20, 'Marco': 23}
new_student = {19:'Karl'}
students.update()
print(students)

# prints: {'Ana': 17, 'Mel': 18, 'Karl': 20, 'Marco': 23}
```

values()

This) method returns a view object that displays a list of all the values in the dictionary.

Example:

```
students = {'Ana':17, 'Mel':18, 'Karl':20, 'Marco': 23}
print(students.values())

# prints: dict_values([17, 18, 20, 23])
```

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

add()		
clear()		
copy()		
difference()		
intersection()		
issubset()		
issuperset()		
pop()		
remove()		
symmetric_diff er ence()		
union()		

update()		

`add()`

This method adds a given element intto a *set*.

Example:

students = {'Ana', 'Mel', 'Karl', 'Marco'}

```
students.add('Natalie')
print(students)

#prints: {'Marco', 'Ana', 'Karl', 'Natalie', 'Mel'}
```

clear()

This method removes all elements from the set.

Example:

```
students = {'Ana', 'Mel', 'Karl', 'Marco'}
students.clear()
print('students:', students)

#pritns: students: set()
```

difference()

This method returns a set that contains the difference between two sets.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Banana', 'Kiwi', 'Papaya', 'Grapes'}
print(fruits.difference(fruits_2))

#prints: {'Melon', 'Apple'}
```

intersection()

This method returns a set containing similarities between two or more sets.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Banana', 'Kiwi', 'Papaya', 'Grapes'}
print(fruits.intersection(fruits_2))

#prints: {'Kiwi', 'Grapes'}
```

issubset()

This method returns True if all items exist in the specified *set*. Otherwise, it will return False.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Banana', 'Kiwi', 'Papaya', 'Grapes', 'Watermelon'}
print(fruits.issubset(fruits_2))

#prints: False
```

issuperset()

This method returns True if a set has all elements of the other set (passed as an argument). Otherwise, it returns False.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Banana', 'Kiwi', 'Papaya', 'Grapes', 'Watermelon'}
print(fruits.issuperset(fruits_2))

#prints: False
```

pop()

This method removes and returns a random element from the set.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
print('Fruits:', fruits.pop())

#prints: Kiwi
```

remove()

This method removes the specified element from a set.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits.remove('Melon')
print('Fruits:', fruits)

#prints: Fruits: {'Apple', 'Grapes', 'Kiwi'}
```

symmetric_difference()

This method returns a *set* containing all items from both *sets* but not the intersection.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Apple', 'Tangerine', 'Kiwi', 'Grapes', 'Watermelon'}
print(fruits.symmetric_difference(fruits_2))

#prints: {'Melon', 'Watermelon', 'Tangerine'}
```

union()

This method returns a set containing all items from the original set and all from the specified set or sets.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = {'Apple', 'Tangerine', 'Kiwi', 'Grapes', 'Watermelon'}
print(fruits.union(fruits_2))

#prints: {'Tangerine', 'Melon', 'Grapes', 'Apple', 'Kiwi', 'Watermelon'}
```

update()

This method updates the original *set* by adding items from all the defined *sets* with no duplicates.

Example:

```
fruits = {'Apple', 'Melon', 'Kiwi', 'Grapes'}
fruits_2 = ['Blackberry', 'Cherry']
fruits.update(fruits_2)
print(fruits)

#prints: {'Kiwi', 'Apple', 'Melon', 'Cherry', 'Grapes', 'Blackberry'}
```

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

read()		
readline()		
readlines()		
write()		
writelines()		

read()

This method returns the specified number of bytes from the file. Default is -1, which means the complete file.

Example:

```
f = open("test.txt", "r")
print(f.read(3))

#prints: Thi
```

readline()

This method returns one line from the file. It is also possible to specify how many bytes from the line to return by using the size parameter

Example:

```
f = open("test.txt", "r")
print(f.readline(20))

#prints: This is a test.
```

readlines()

This method returns a list containing individually lines in the file as a list item. It is possible to

use the hint parameter to limit the number of lines returned.

Example:

```
f = open("pokemons.txt", "r")
print(f.readlines())

# prints: ['bulbasaur\n', 'ivysaur\n', 'venusaur\n', 'charmander\n', 'charmeleon\n', 'charizard\n']
```

write()

This method writes a specified text to the file and returns the number of characters written.

Example:

```
f = open("test.txt", "w")
new_word = f.write('Cambio.')
print(new_word)
f.close()

# prints:7
```

writelines()

This method writes a sequence of strings to the file. The sequence can be any iterable object producing strings, commonly a list of strings. There is no return value.

Example:

```
f = open("test.txt", "a")
f.writelines(['test\n', 'test\n'])
f.close()
f = open("test.txt", "r")
print(f.read())

# prints:Cambio.test
test
```