THEORY QUESTIONS ASSIGNMENT
OLEKSANDRA YANKOVSKA

1. What is Python and what are its main features?
Python is one of the top languages for data analysts and data scientists. Its simple syntax makes it popular for data wrangling, and it also has a multitude of libraries that enable data visualization, data analysis, and machine learning.

Its main features:
Features in Python,
Free and Open Source,
Easy to code,
Easy to Read,
Object-Oriented Language,
GUI Programming Support,
High-Level Language,
Extensible feature,
Easy to Debug.

2. Discuss the difference between Python 2 and Python 3
Python 3 has an easier syntax compared to Python 2. A lot of libraries of Python 2 are not forward compatible. A lot of libraries are created in Python 3 to be strictly used with Python 3. Python 2 is no longer in use since 2020.

3. What is PEP 8?
PEP 8 is a coding convention, a set of recommendations, about how to write your Python code more readable.

4. In computing / computer science what is a program?
In computing, a program is a specific set of ordered operations for a computer to perform. The program contains a one-at-a-time sequence of instructions that the computer follows. Typically, the program is put into a storage area accessible to the computer.

5. In computing / computer science what is a process?
In computing, a process is the instance of a computer program that is being executed by one or many threads. There are many different process models, some of which are light weight, but almost all processes (e.g. entire virtual machines) are rooted in an operating system (OS) process which comprises the program code, assigned system resources, physical and logical access permissions, and data structures to initiate, control and coordinate execution activity.
While a computer program is a passive collection of instructions typically stored in a file on disk, a process is the execution of those instructions after being loaded from the disk into memory.

6. In computing / computer science what is cache?

A cache is hardware or software that is used to store something, usually data, temporarily in a computing environment. It is a small amount of faster, more expensive memory used to improve the performance of recently or frequently accessed data.

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

With computer programming, a thread is a small set of instructions designed to be scheduled and executed by the CPU independently of the parent process.

Multithreading is when a program creates multiple threads with execution cycling among them, so one longer-running task doesn't block all the others. This works well for tasks that can be broken down into smaller subtasks, which can then each be given to a thread to be completed.

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

Concurrency is when two or more tasks can start, run, and complete in overlapping time periods. It doesn't necessarily mean they'll ever both be running at the same instant. For example, multitasking on a single-core machine. Parallelism is when tasks literally run at the same time, e.g., on a multicore processor.

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

Python Global Interpreter Lock (GIL) is a type of process lock which is used by python whenever it deals with processes. Generally, Python only uses only one thread to execute the set of written statements. This means that in python only one thread will be executed at a time. The performance of the single-threaded process and the multi-threaded process will be the same in python and this is because of GIL in python. We can not achieve multithreading in python because we have global interpreter lock which restricts the threads and works as a single thread.

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

1. DRY – Don't Repeat Yourself

DRY refers to code writing methodology. DRY usually refers to code duplication. If we write the same logic more than once, we should "DRY up our code."

A common way to DRY up code is to wrap our duplicated logic with a function and replace all places it appears with function calls.

KISS – Keep It Simple Stupid

Keeping it simple surpsingly hard. Usually when someone tries to over-engineer a soluiton to a problem. For example an Architect suggests

creating a Microservice framework for a simple website. The Engineer will then say: “Let’s KISS it and do something simpler”.

Another rule of thumb is whenever you think to yourself “Finally I’m going to use something from my Computer Science degree,” you should probably KISS it.

BDUF – Big Design Up Front
This is a relic from the waterfall era before everyone became cool and Agile.

This acronym is here to remind us not get over carried with super complex architecture. We shouldn’t spend 3 months designing our application before even writing the first line of code. Start small and iterate.

11. What is a Garbage Collector in Python and how does it work?
The garbage collector is keeping track of all objects in memory. A new object starts its life in the first generation of the garbage collector. If Python executes a garbage collection process on a generation and an object survives, it moves up into a second, older generation.

12. How is memory managed in Python?
The Python memory is primarily managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have access to this private heap and interpreter takes care of this Python private heap.

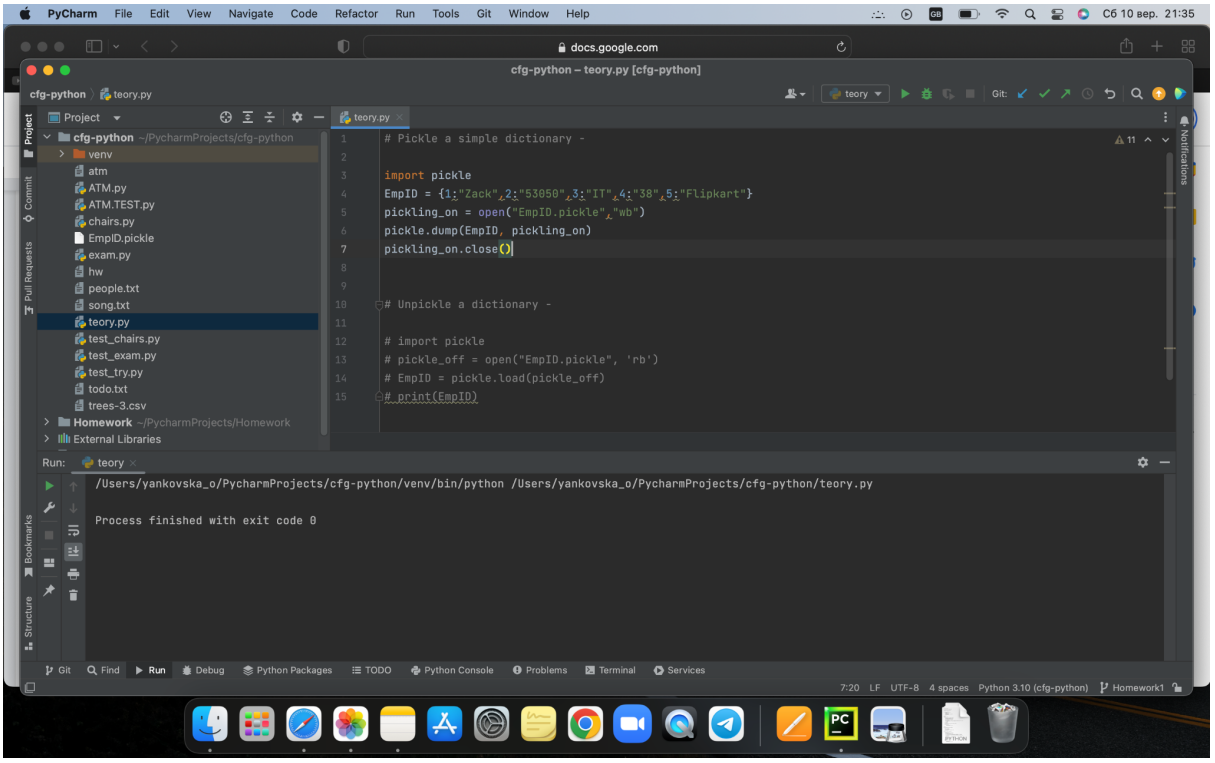
13. What is a Python module?
A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use.

14. What is docstring in Python?
A Python docstring is a string used to document a Python module, class, function or method, so programmers can understand what it does without having to read the details of the implementation. Also, it is a common practice to generate online (html) documentation automatically from docstrings.

15. What is pickling and unpickling in Python? Example usage.
“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy.

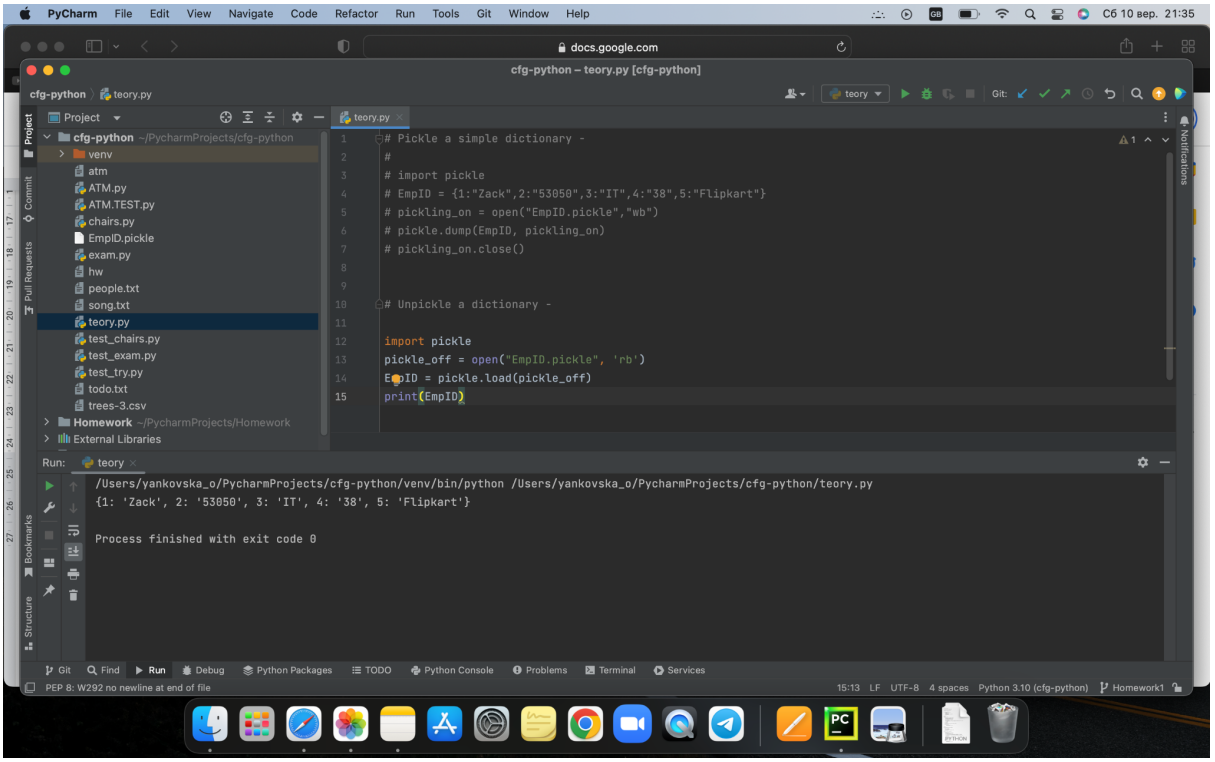
Pickle a simple dictionary –

```
import pickle
EmpID = {1:"Zack",2:"53050",3:"IT",4:"38",5:"Flipkart"}
pickling_on = open("EmpID.pickle","wb")
pickle.dump(EmpID, pickling_on)
pickling_on.close()
```



Unpickle a dictionary –

```
import pickle
pickle_off = open("EmpID.pickle", 'rb')
EmpID = pickle.load(pickle_off)
print(EmpID)
```



16. What are the tools that help to find bugs or perform static analysis?
Pychecker and Pylint are the static analysis tools that help to find bugs in python.

Pychecker is an opensource tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.

Pylint is highly configurable and it acts like special programs to control warnings and errors, it is an extensive configuration file Pylint is also an opensource tool for static code analysis it looks for programming errors and is used for coding standard. it checks the length of each programming line. it checks the variable names according to the project style. it can also be used as a standalone program, it also integrates with python IDEs such as Pycharm, Spyder, Eclipse, and Jupyter

Pychecker can be simply installed by using pip package pip install Pychecker if suppose if you use python 3.6 version use upgrade pip install Pychecker —upgrade Pylint can be simply installed by using pip package

pip install Pylint

if suppose if you use python 3.6 version use upgrade

pip install Pylint —upgrade

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

example.
Pass by reference means that you have to pass the function(reference) to a variable which refers that the variable already exists in memory.

Here, the variable(the bucket) is passed into the function directly. The variable acts as a Package that comes with its contents(the objects).

```
def same list(list):
return list
mylist=["X"
same_list(my_list)
print(my_list)
```

In the above code image both “list” and “my_list” are the same container variable and therefore refer to the exact same object in the memory. Any operation performed by the function on the variable or the object will be directly reflected by the function caller. For instance, the function could completely change the variable’s content, and point it at a completely different object:

```
def set _list(list):
list=["A"]
return list
mylist=["X"]
same_list(my_list)
print(my_list)
```

To summarize, in pass-by-reference the function and the caller use the same variable and object.

What is Pass by Value In Python?
In this approach, we pass a copy of actual variables in function as a parameter. Hence any modification on parameters inside the function will not reflect in the actual variable.

```
def same_list(list):
return list
mylist=["X"]
same_list(my_list)
print(my_list)
```

The same is true for any operation performed by the function on the variable or the object

```
def add (list):
list.append!"B"]
```

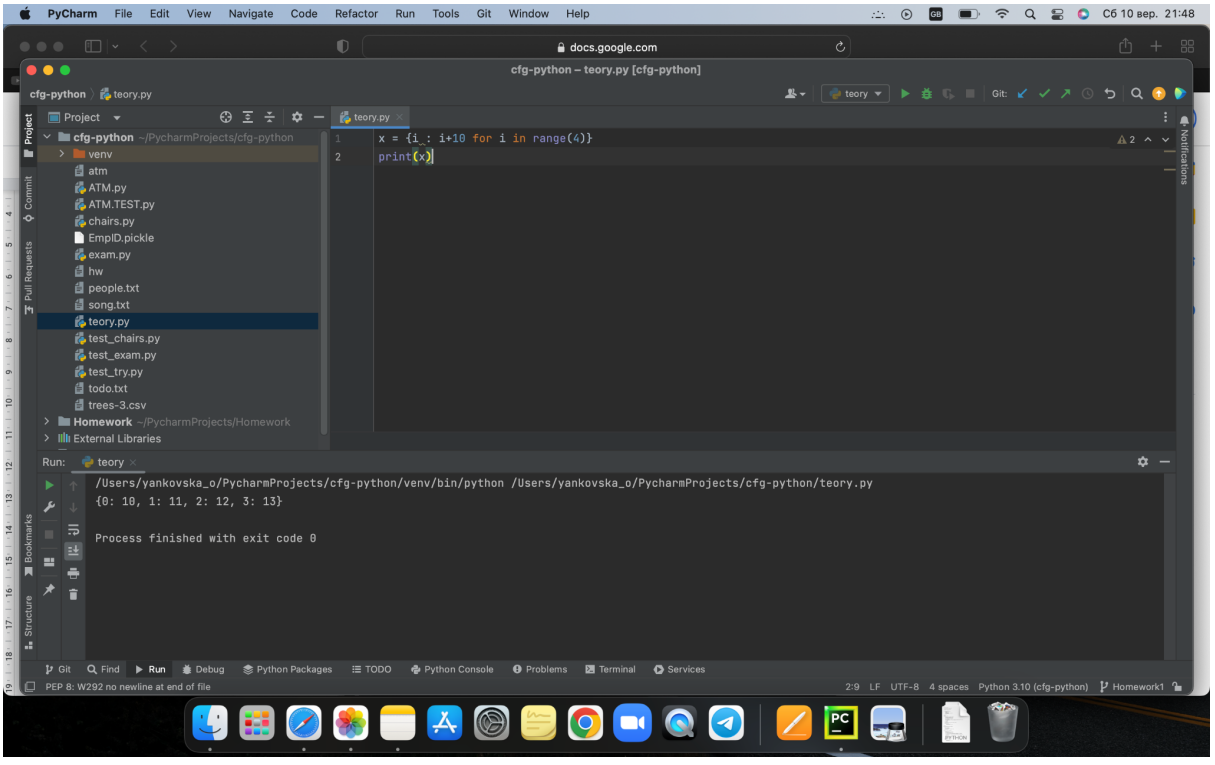
return list
mylist=["X"]
add(my_list)
print(my_list)

To summarize the copies of the variables and the objects in the context of the caller of the function are completely isolated.

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

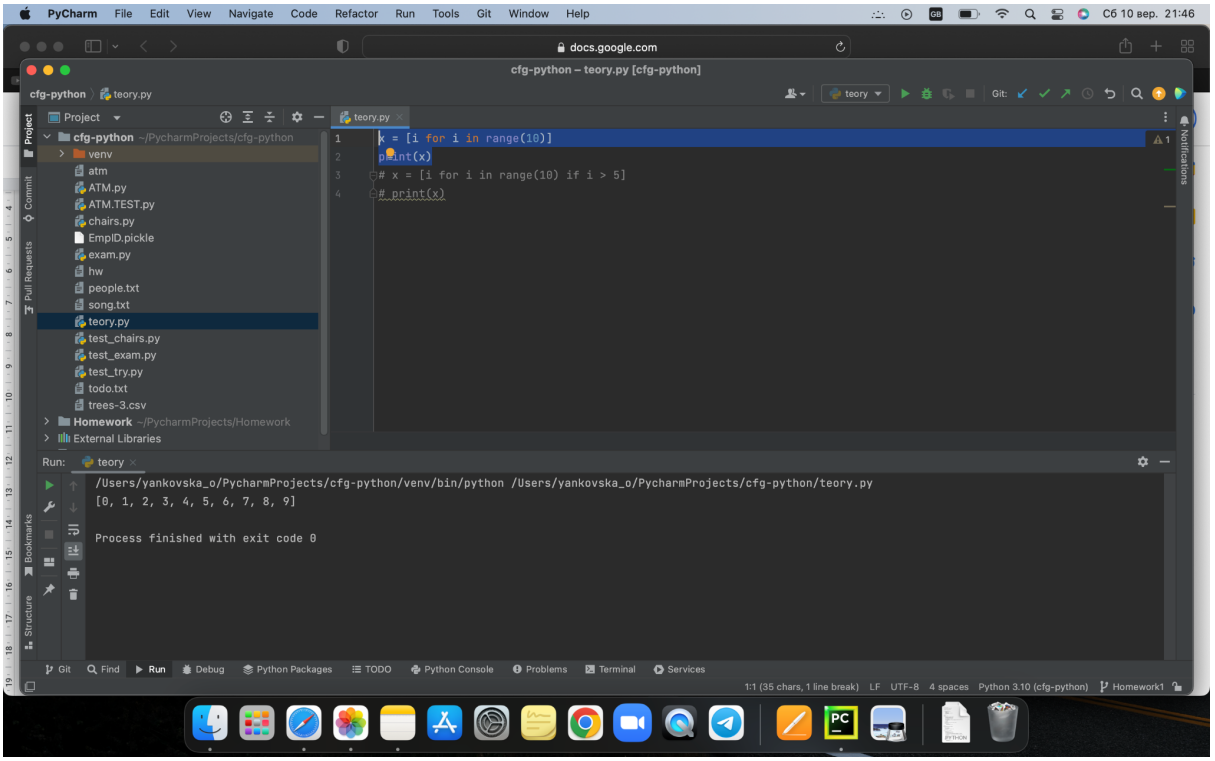
Dictionary comprehension is one way to create a dictionary in Python. It creates a dictionary by merging two sets of data which are in the form of either lists or arrays.

x = {i : i+10 for i in range(4)}
print(x)



List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list. Example: Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

x = [i for i in range(10)]
print(x)



19. What is namespace in Python?

A namespace is a collection of currently defined symbolic names along with information about the object that each name references. You can think of a namespace as a dictionary in which the keys are the object names and the values are the objects themselves.

20.What is pass in Python?

The pass statement is used as a placeholder for future code.

When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

Empty code is not allowed in loops, function definitions, class definitions, or in if statements.

21. What is unit test in Python?

Unit testing is a method for testing software that looks at the smallest testable pieces of code, called units, which are tested for correct operation. By doing unit testing, we can verify that each part of the code, including helper functions that may not be exposed to the user, works correctly and as intended.

The idea is that we are independently checking each small piece of our program to ensure that it works. This contrasts with regression and integration testing, which tests that the different parts of the program work well together and as intended.

22. In Python what is slicing?

The slice() function returns a slice object. A slice object is used to specify how to slice a sequence. You can specify where to start the slicing, and where to end. You can also specify the step, which allows you to e.g. slice only every other item.

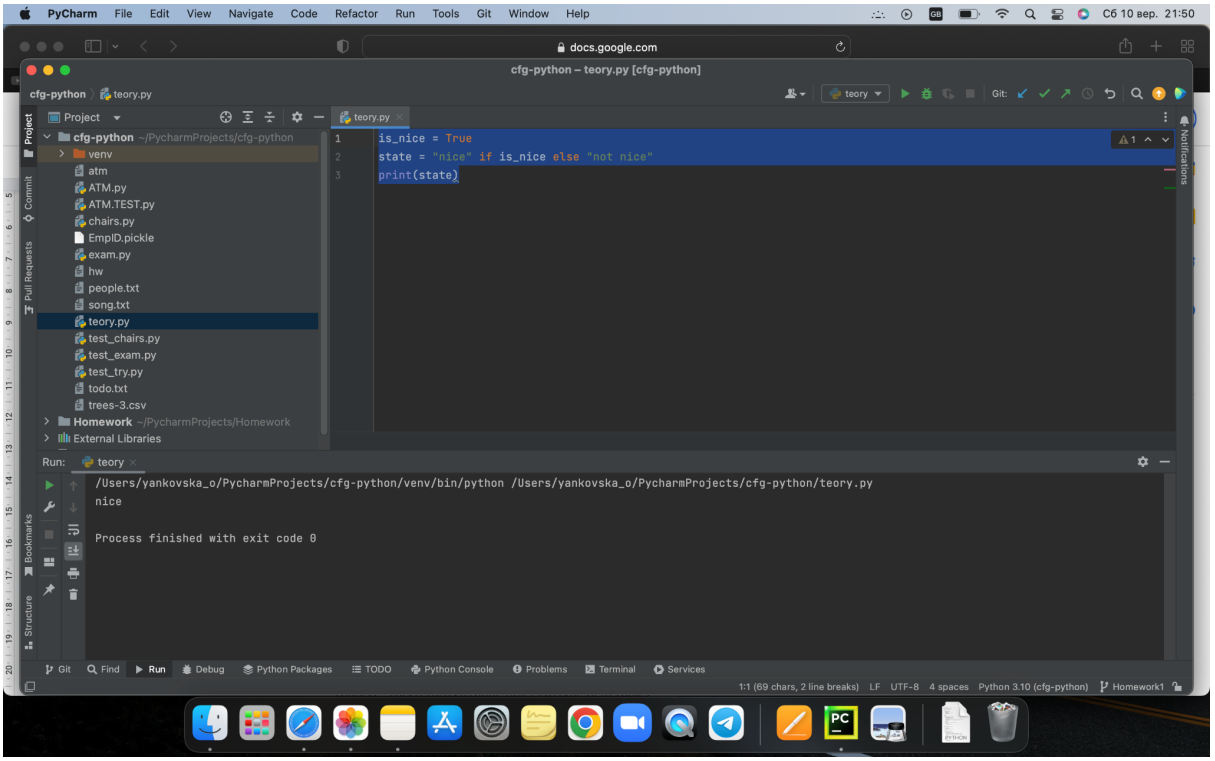
23. What is a negative index in Python?

Negative indexes are a way to allow you to index into a list, tuple or other indexable container relative to the end of the container, rather than the start. The reason this is called negative indexing is that the number you are using is less than the 0th element in the list.

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

Many programming languages support ternary operator, which basically define a conditional expression. Similarly the ternary operator in python is used to return a value based on the result of a binary condition. It takes binary value(condition) as an input, so it looks similar to an “if-else” condition block.

```
is_nice = True
state = "nice" if is_nice else "not nice"
print(state)
```



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

The special syntax *args in function definitions in python is used to pass a variable number of arguments to a function. It is used to pass a non-key worded, variable-length argument list.

The special syntax **kwargs in function definitions in python is used to pass a keyworded, variable-length argument list. We use the name kwargs with the double star. The reason is that the double star allows us to pass through keyword arguments (and any number of them).

26. How are range and xrange different from one another?
The range() and xrange() are two functions that could be used to iterate a certain number of times in for loops in Python. In Python 3, there is no xrange, but the range function behaves like xrange in Python 2. If you want to write code that will run on both Python 2 and Python 3, you should use range().

range() – This returns a range object (a type of iterable).
xrange() – This function returns the generator object that can be used to display numbers only by looping. The only particular range is displayed on demand and hence called "lazy evaluation".
Both are implemented in different ways and have different characteristics associated with them. The points of comparison are:

Return Type
Memory
Operation Usage
Speed

27. What is Flask and what can we use it for?
Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file

28. What are clustered and non-clustered index in a relational database?
1. Clustered Index :
Clustered index is created only when both the following conditions satisfy –

The data or file, that you are moving into secondary memory should be in sequential or sorted order.
There should be a key value, meaning it can not have repeated values.

Whenever you apply clustered indexing in a table, it will perform sorting in that table only. You can create only one clustered index in a table like primary key. Clustered index is as same as dictionary where the data is arranged by alphabetical order.

Non-Clustered Index is similar to the index of a book. The index of a book consists of a chapter name and page number, if you want to read any

topic or chapter then you can directly go to that page by using index of that book. No need to go through each and every page of a book.

The data is stored in one place, and index is stored in another place. Since, the data and non-clustered index is stored separately, then you can have multiple non-clustered index in a table.

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

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks. For example, Transaction A might hold a lock on some rows in the Accounts table and needs to update some rows in the Orders table to finish.

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

A Live lock is one, where a request for exclusive lock is denied continuously because a series of overlapping shared locks keeps on interfering each other and to adapt from each other they keep on changing the status which further prevents them to complete the task.

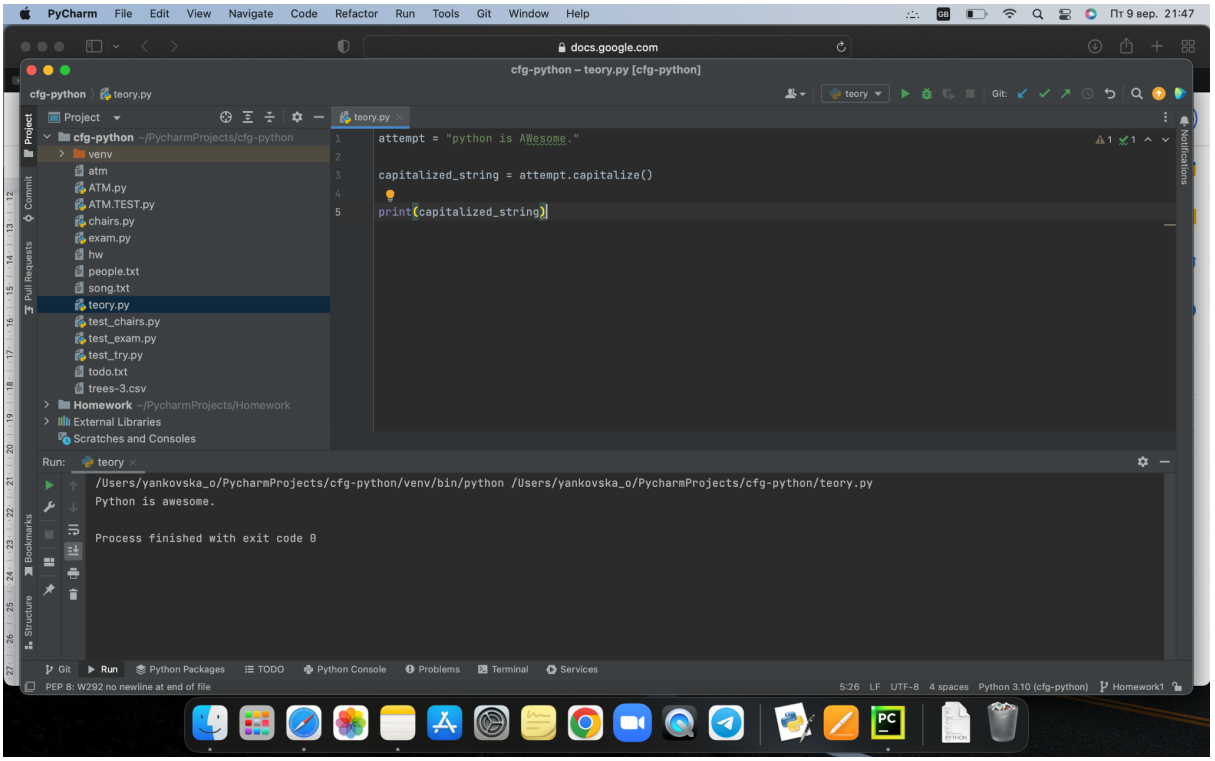
2. Python string methods: describe each method and provide an example

capitalize()
The capitalize() method converts the first character of a string to an uppercase letter and all other alphabets to lowercase. The capitalize() method returns a new string and doesn't modify the original string.

The syntax of the capitalize() method is:
string.capitalize()

EXAMPLE:

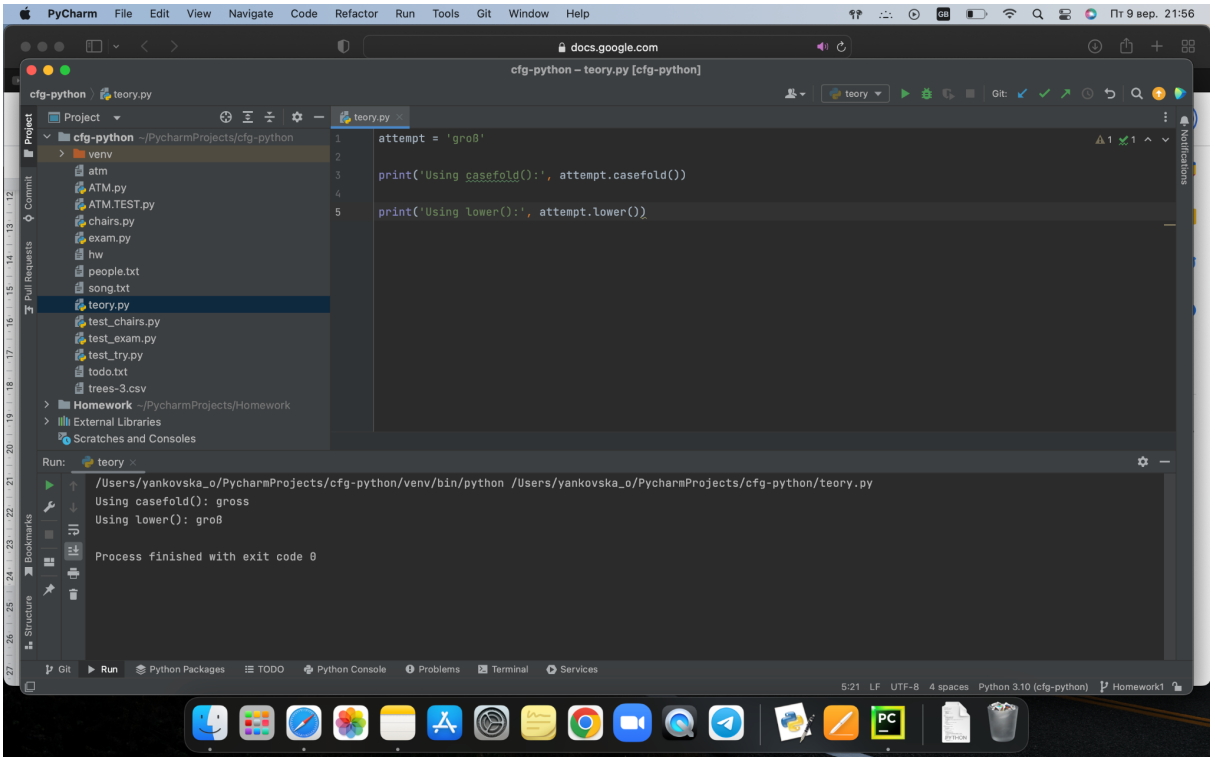
```
attempt = "python is AWesome."
capitalized_string = attempt.capitalize()
print(capitalized_string)
```



---

casefold()
The casefold() method converts all characters of the string into lowercase letters and returns a new string.
The syntax of the casefold() method is:
str.casefold()

EXAMPLE:
attempt = 'groß'
print('Using casefold():', attempt.casefold())
print('Using lower():', attempt.lower())

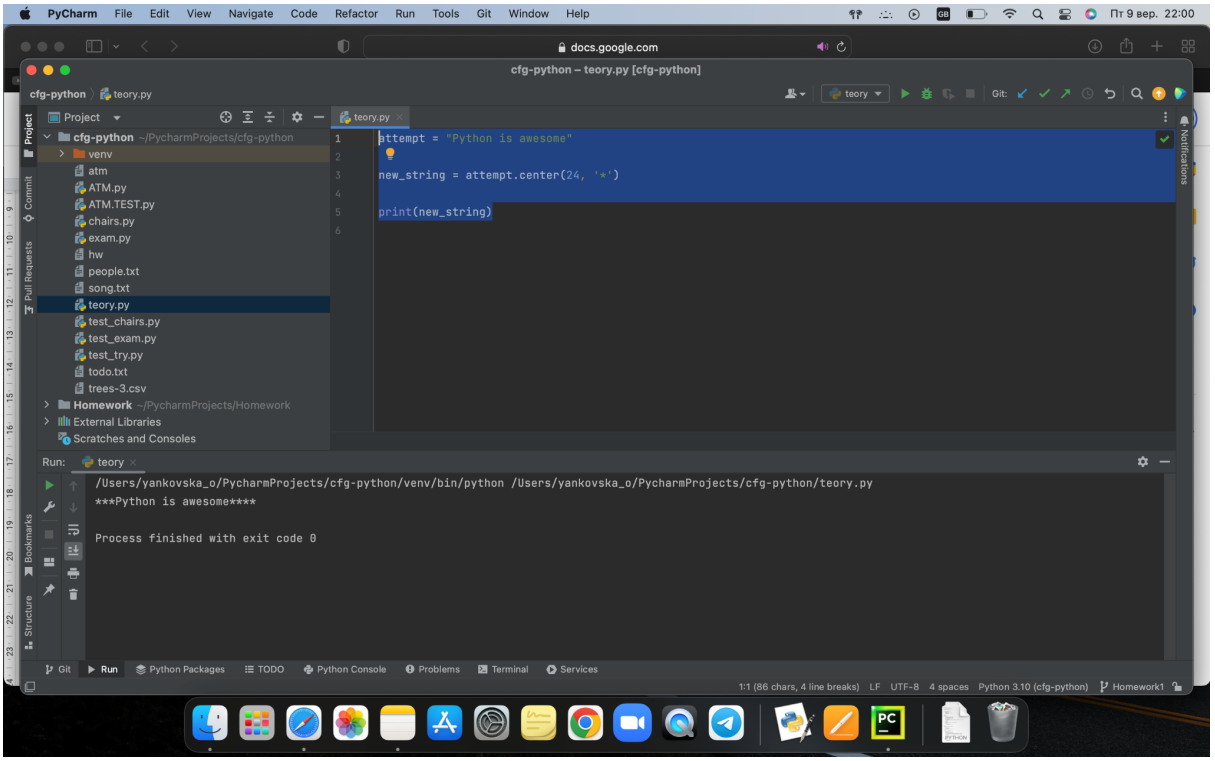


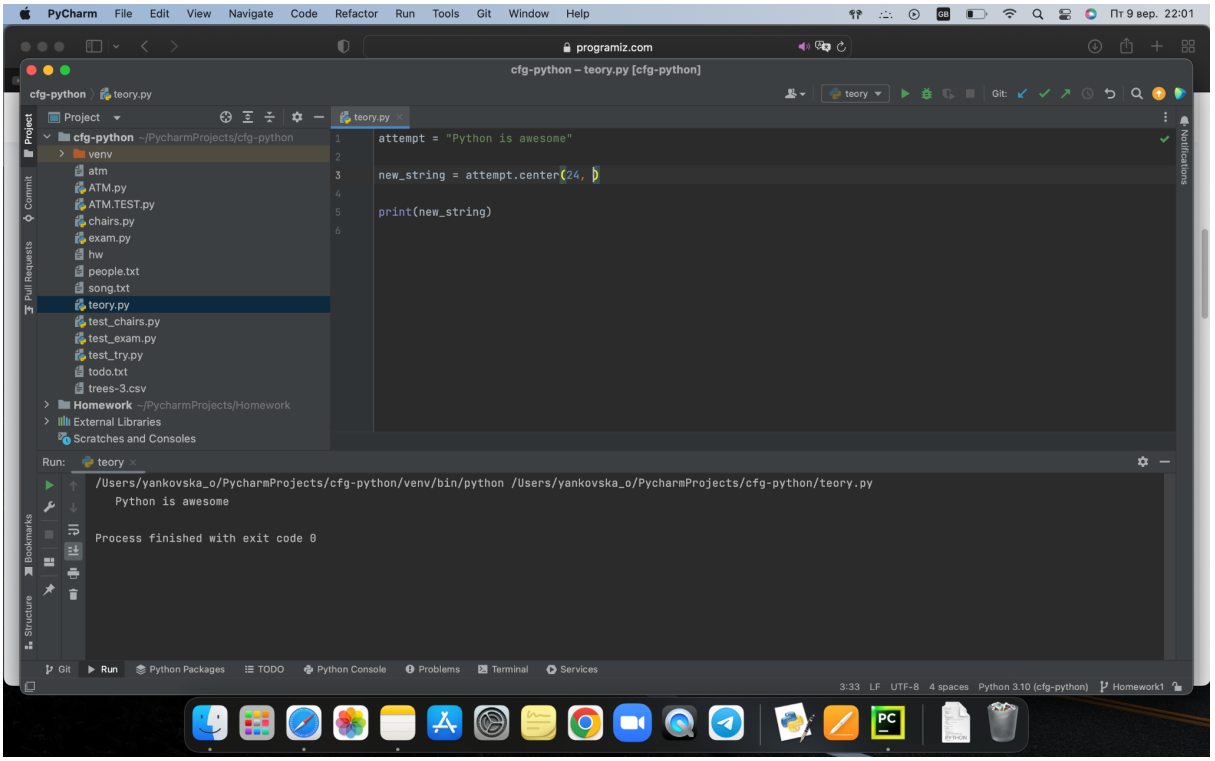
---

center()
The center() method returns a new centered string after padding it with the specified character. The center() method doesn't modify the original string.
The syntax of the center() method is:
str.center(width, [fillchar])

The center() method takes two parameters:
width - length of the string with padded characters
fillchar (optional) - padding character (If fillchar is not provided, whitespace is taken as the default argument.)

EXAMPLE:
attempt = "Python is awesome"
new_string = attempt.center(24, "*")
print(new_string)





count()
The count() method returns the number of occurrences of a substring in the given string.
The syntax of count() method is:
string.count(substring, start=..., end=...)

count() method only requires a single parameter for execution.

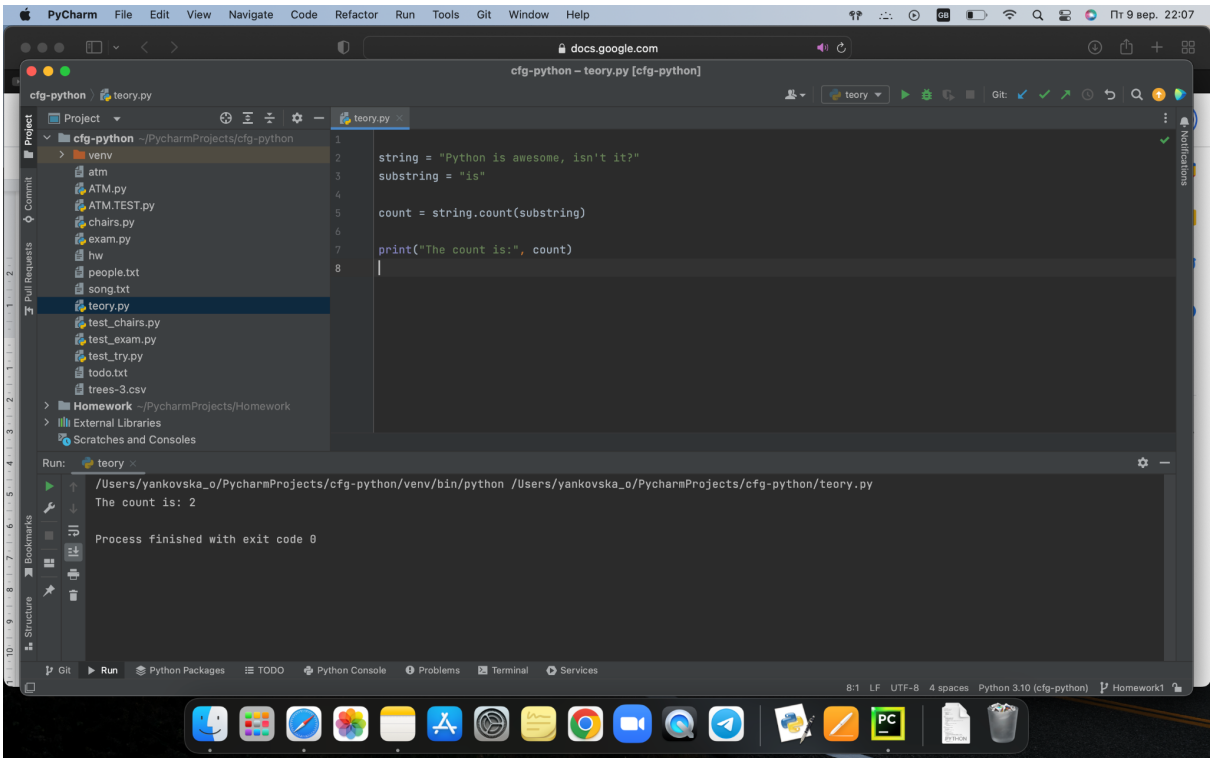
However, it also has two optional parameters:
substring - string whose count is to be found.
start (Optional) - starting index within the string where search starts.
end (Optional) - ending index within the string where search ends.

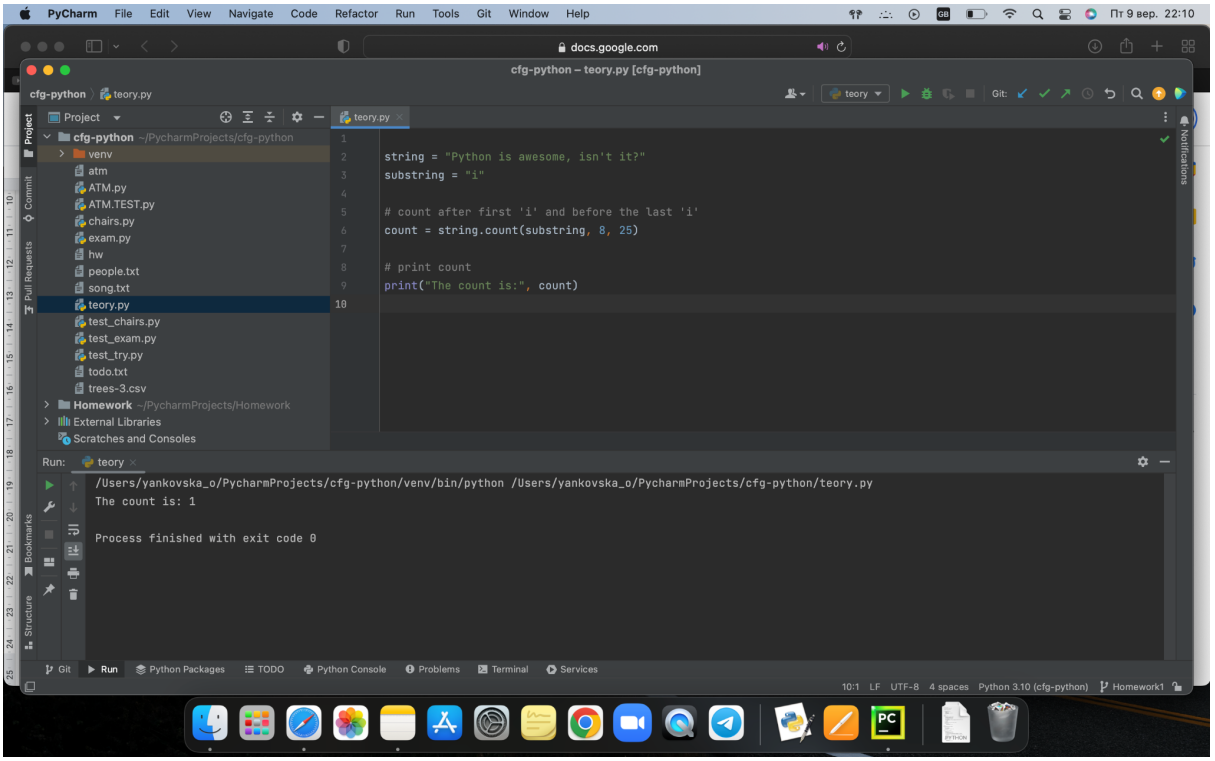
EXAMPLE:
string = "Python is awesome, isn't it?"
substring = "is"

count = string.count(substring)
print("The count is:", count)

---
string = "Python is awesome, isn't it?"
substring = "i"

count = string.count(substring, 8, 25)
print("The count is:", count)





---

endswith()

The endswith() method returns True if a string ends with the specified suffix. If not, it returns False.

The syntax of endswith() is:

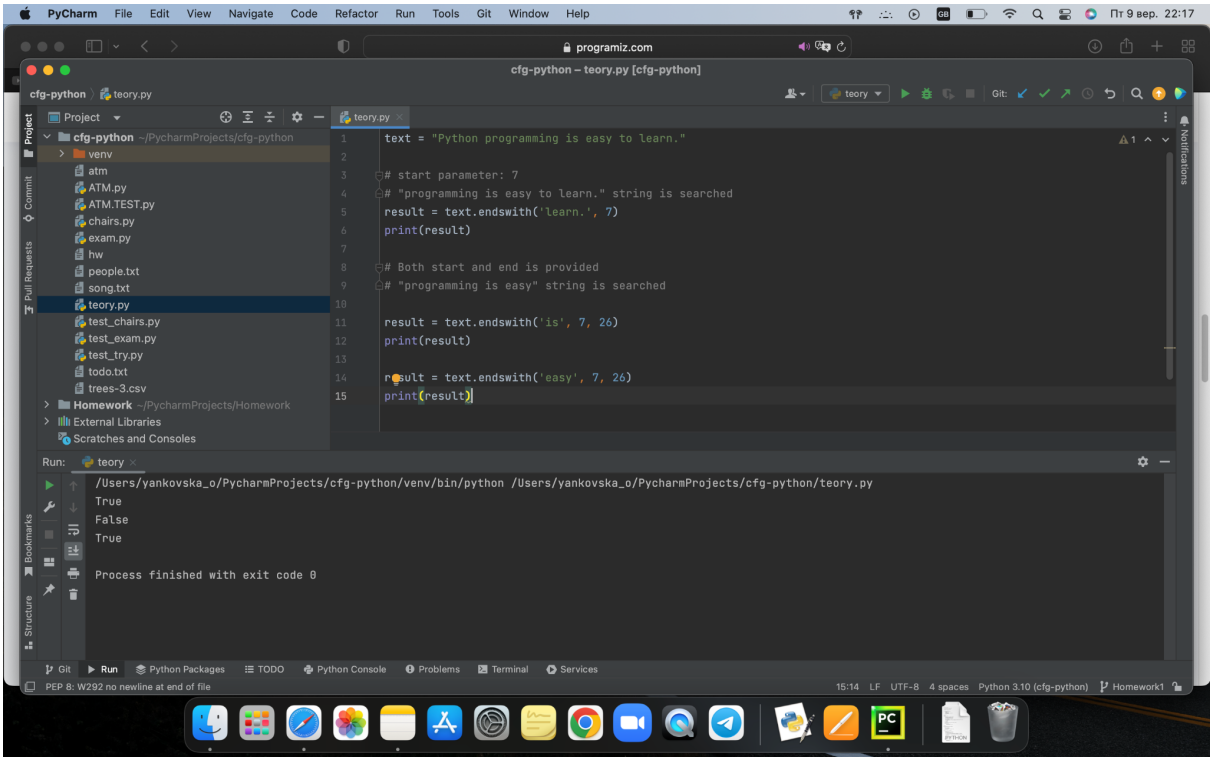
str.endswith(suffix[, start[, end]])

The endswith() takes three parameters:

suffix - String or tuple of suffixes to be checked
start (optional) - Beginning position where suffix is to be checked within the string.
end (optional) - Ending position where suffix is to be checked within the string.

EXAMPLE:

```
text = "Python programming is easy to learn."

# start parameter: 7
# "programming is easy to learn." string is searched
result = text.endswith('learn.', 7)
print(result)

# Both start and end is provided
# "programming is easy" string is searched
result = text.endswith('is', 7, 26)
print(result)

result = text.endswith('easy', 7, 26)
print(result)
```

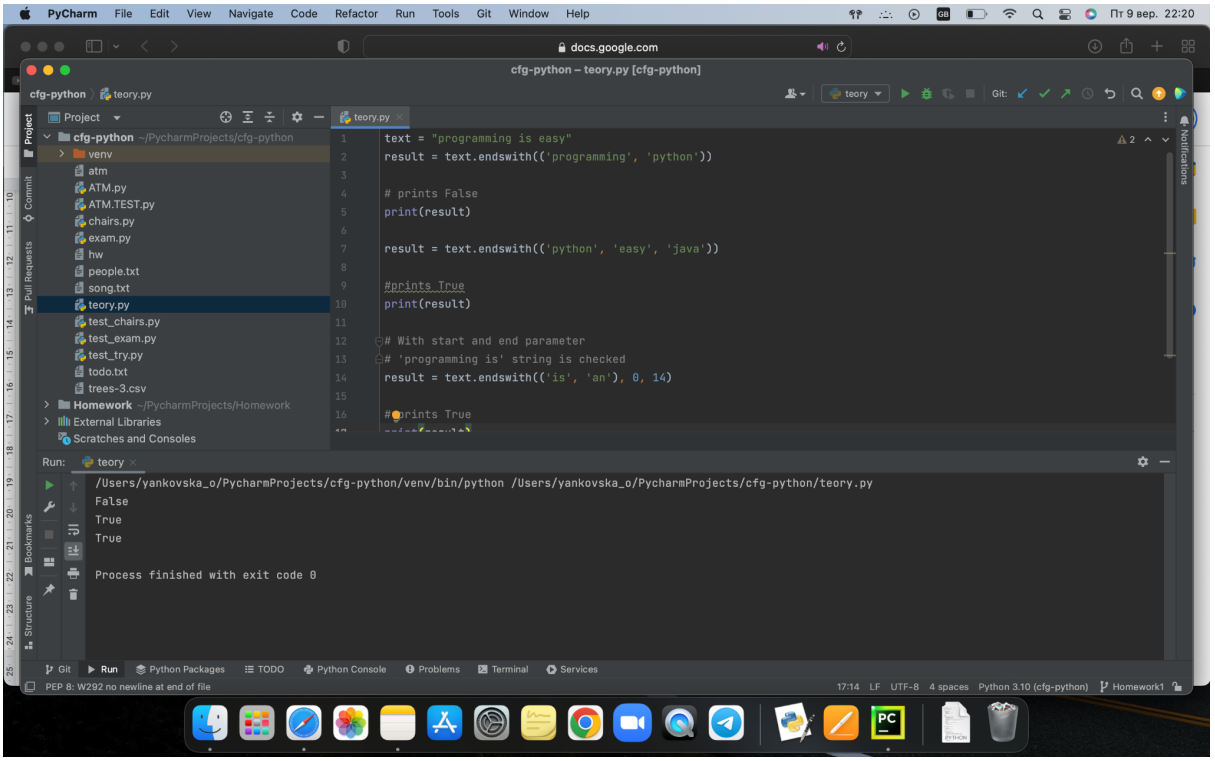


It's possible to pass a tuple suffix to the endswith() method in Python.

If the string ends with any item of the tuple, endswith() returns True. If not, it returns False

EXAMPLE:

```
text = "programming is easy"
result = text.endswith(('programming', 'python'))
print(result)

result = text.endswith(('python', 'easy', 'java'))
print(result)

# 'programming is' string is checked
result = text.endswith(('is', 'an'), 0, 14)
print(result)
```



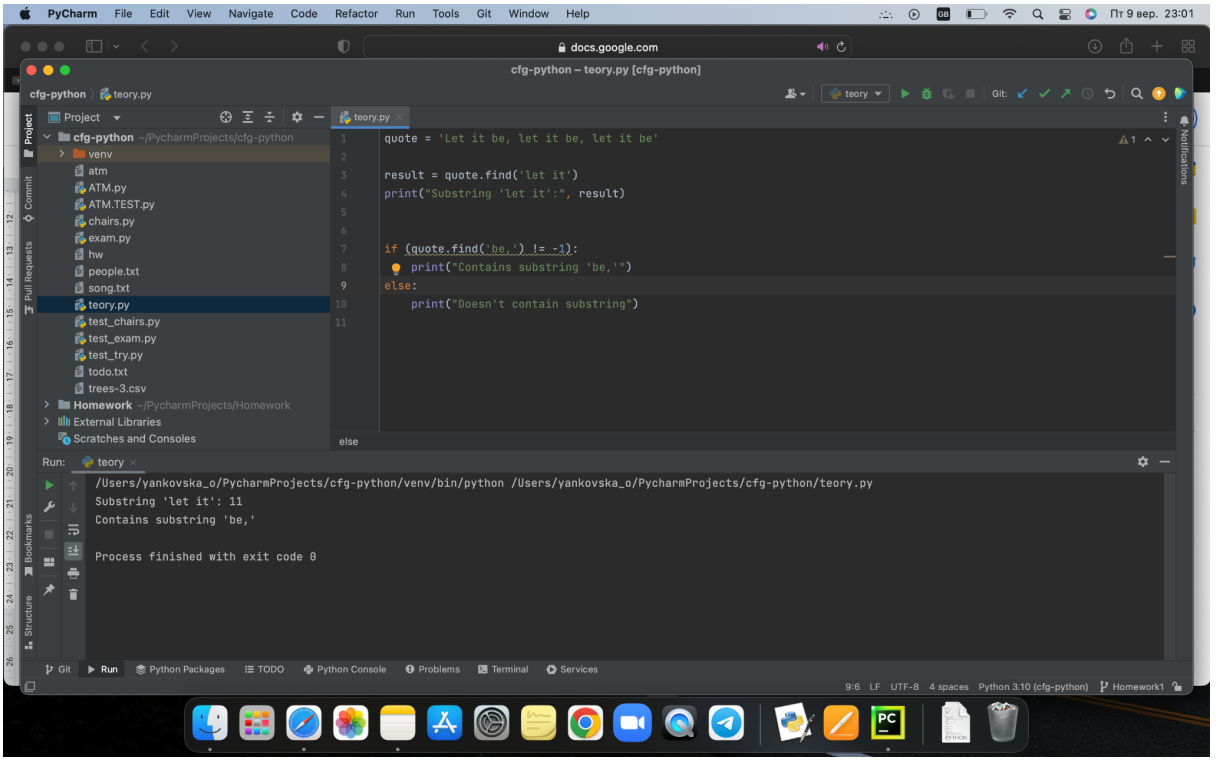
---

find()
The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1. The syntax of the find() method is:
str.find(sub[, start[, end]])

The find() method takes maximum of three parameters:
sub - It is the substring to be searched in the str string.
start and end (optional) - The range str[start:end] within which substring is searched.

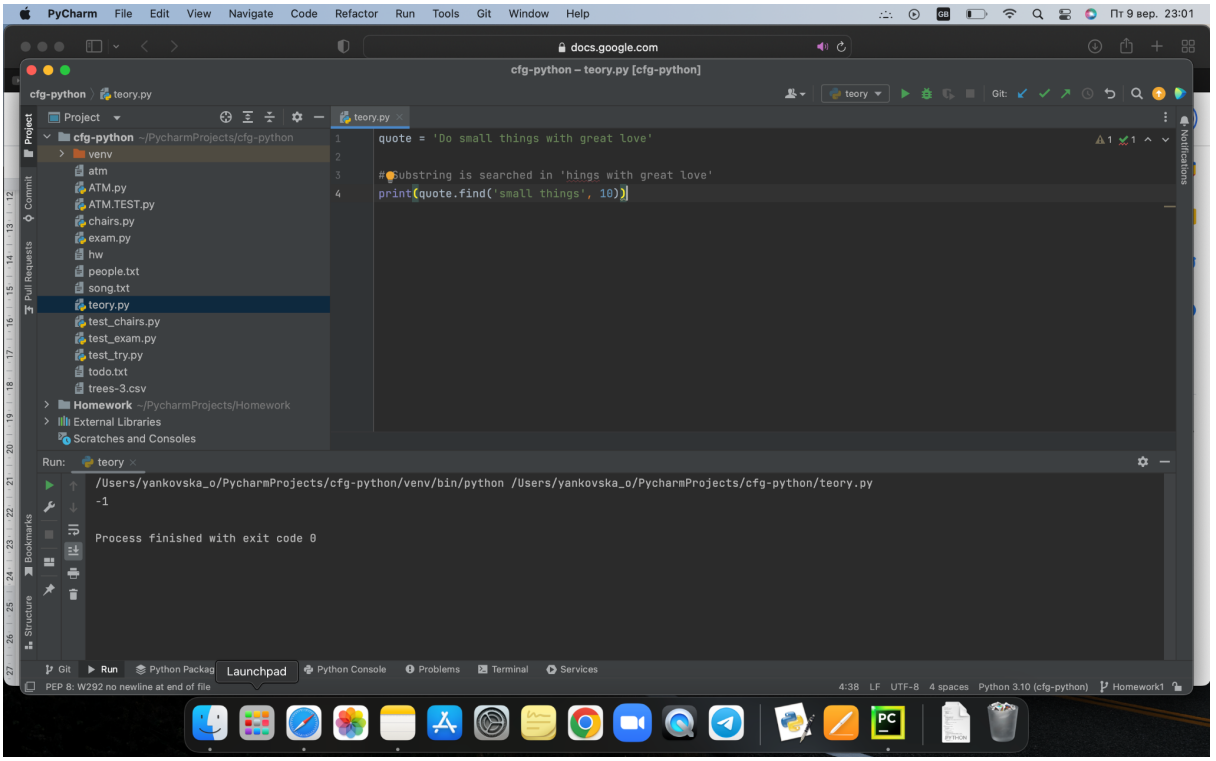
EXAMPLE:

```
quote = 'Let it be, let it be, let it be'

result = quote.find('let it')
print("Substring 'let it':", result)

how to use:
if (quote.find('be,') != -1):
   print("Contains substring 'be,'")
else:
   print("Doesn't contain substring")
```



EXAMPLE2:
quote = 'Do small things with great love'

Substring is searched in 'hings with great love'
print(quote.find('small things', 10))



---

format()
The string format() method formats the given string into a nicer output in Python.

The syntax of the format() method is:

template.format(p0, p1, ..., k0=v0, k1=v1, ...)
Here, p0, p1,... are positional arguments and, k0, k1,... are keyword arguments with values v0, v1,... respectively.

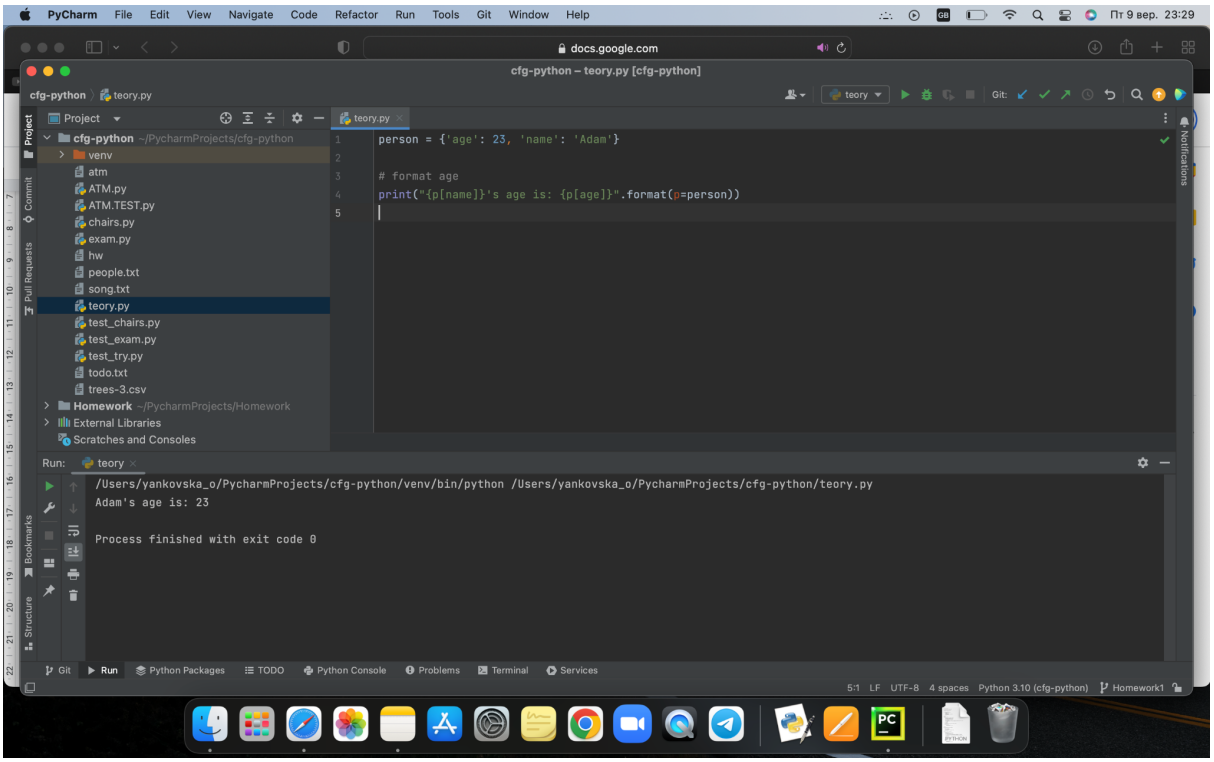
And, template is a mixture of format codes with placeholders for the arguments.

format() method takes any number of parameters. But, is divided into two types of parameters:

Positional parameters - list of parameters that can be accessed with index of parameter inside curly braces {index}
Keyword parameters - list of parameters of type key=value, that can be accessed with key of parameter inside curly braces {key}

EXAMPLE:
person = {'age': 23, 'name': 'Adam'}

# format age
print("{p[name]}'s age is: {p[age]}".format(p=person))



---

index()
The index() method returns the index of a substring inside the string (if found). If the substring is not found, it raises an exception.

It's syntax is:
str.index(sub[, start[, end]])

The index() method takes three parameters:
sub - substring to be searched in the string str.
start and end(optional) - substring is searched within str[start:end]

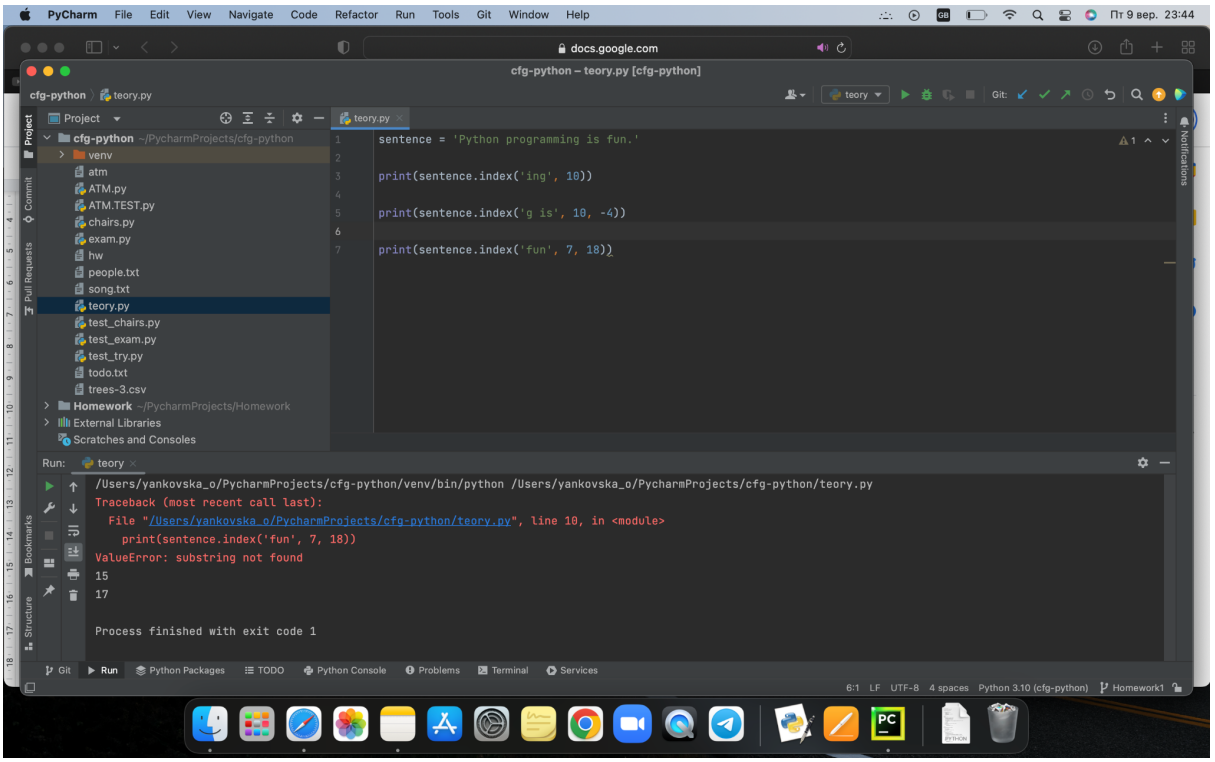
If substring exists inside the string, it returns the lowest index in the string where substring is found.
If substring doesn't exist inside the string, it raises a ValueError exception.
The index() method is similar to the find() method for strings.
The only difference is that find() method returns -1 if the substring is not found, whereas index() throws an exception.

EXAMPLE
sentence = 'Python programming is fun.'

print(sentence.index('ing', 10))

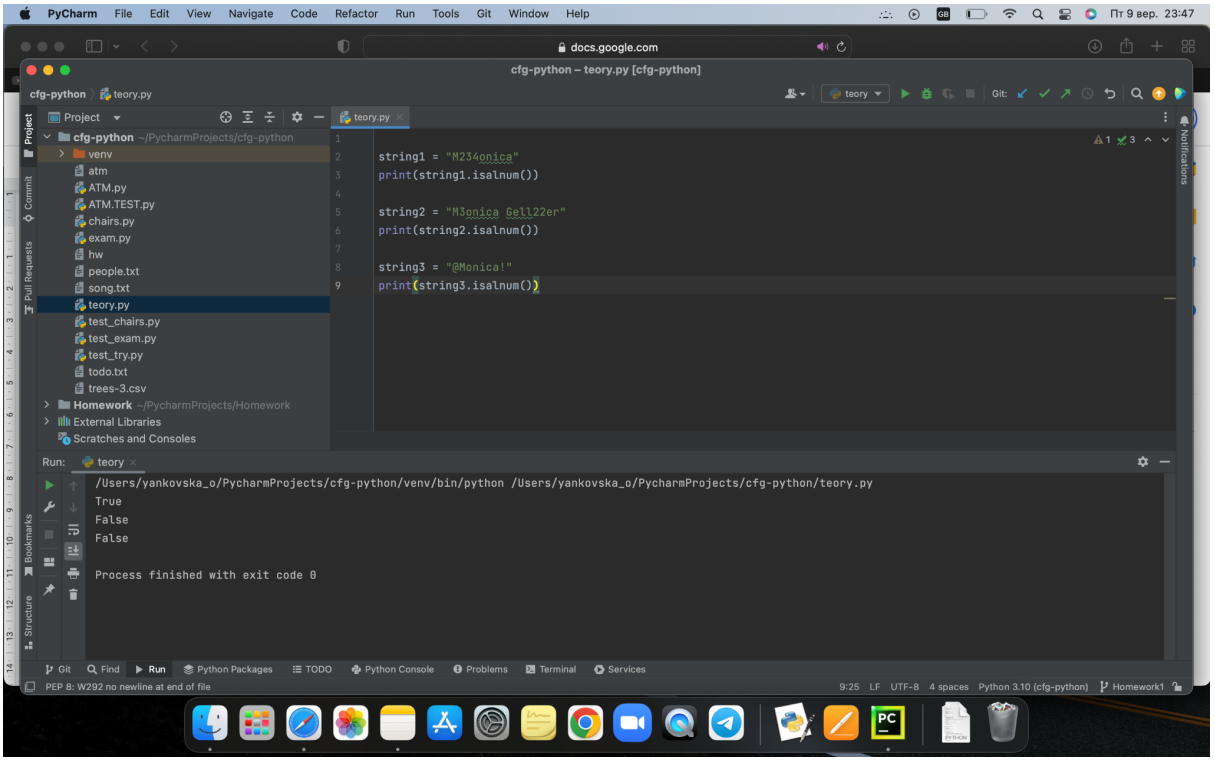
print(sentence.index('g is', 10, -4))

print(sentence.index('fun', 7, 18))



isalnum()
The isalnum() method returns True if all characters in the string are alphanumeric (either alphabets or numbers). If not, it returns False.
The syntax of the isalnum() method is:
string.isalnum()

EXAMPLE:
```
string1 = "M234onica"
print(string1.isalnum())

string2 = "M3onica Gell22er"
print(string2.isalnum())

string3 = "@Monica!"
print(string3.isalnum())
```



---

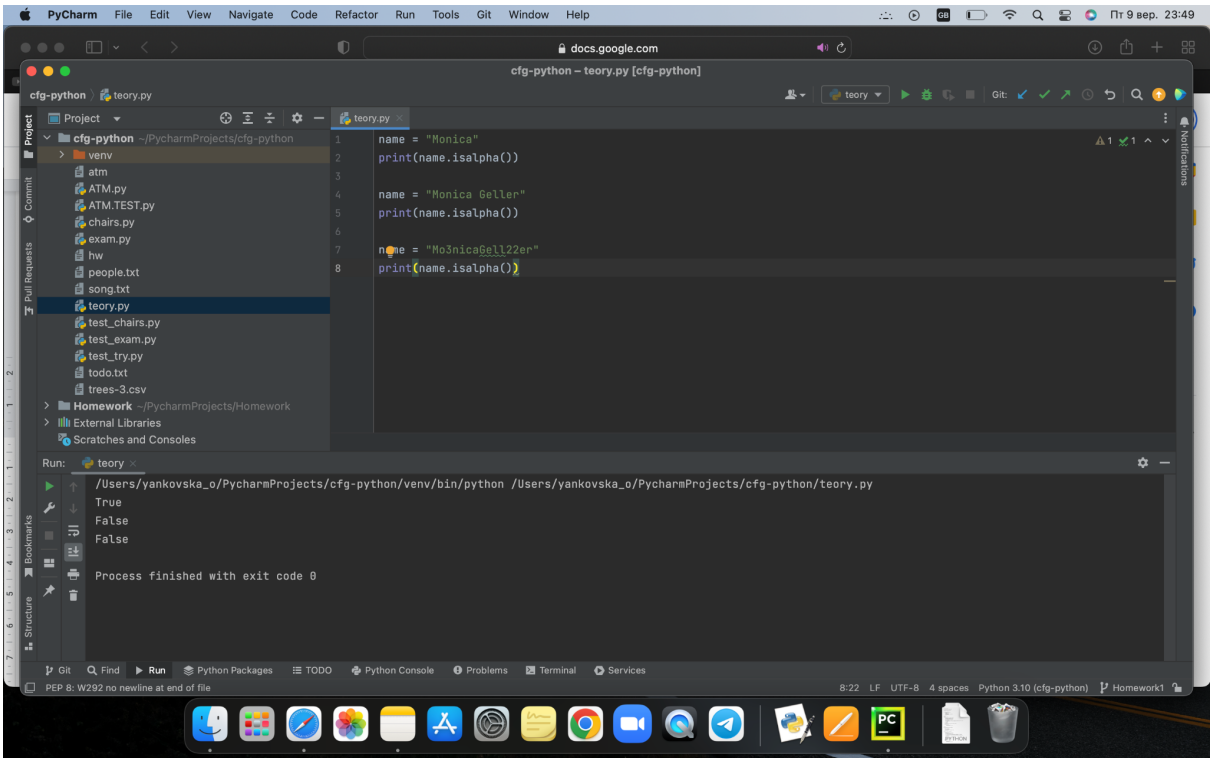
isalpha()
The isalpha() method returns True if all characters in the string are alphabets. If not, it returns False.

The syntax of isalpha() is:
string.isalpha()
False if at least one character is not alphabet.

EXAMPLE:
```
name = "Monica"
print(name.isalpha())

name = "Monica Geller"
```

print(name.isalpha())

name = "Mo3nicaGell22er"
print(name.isalpha())

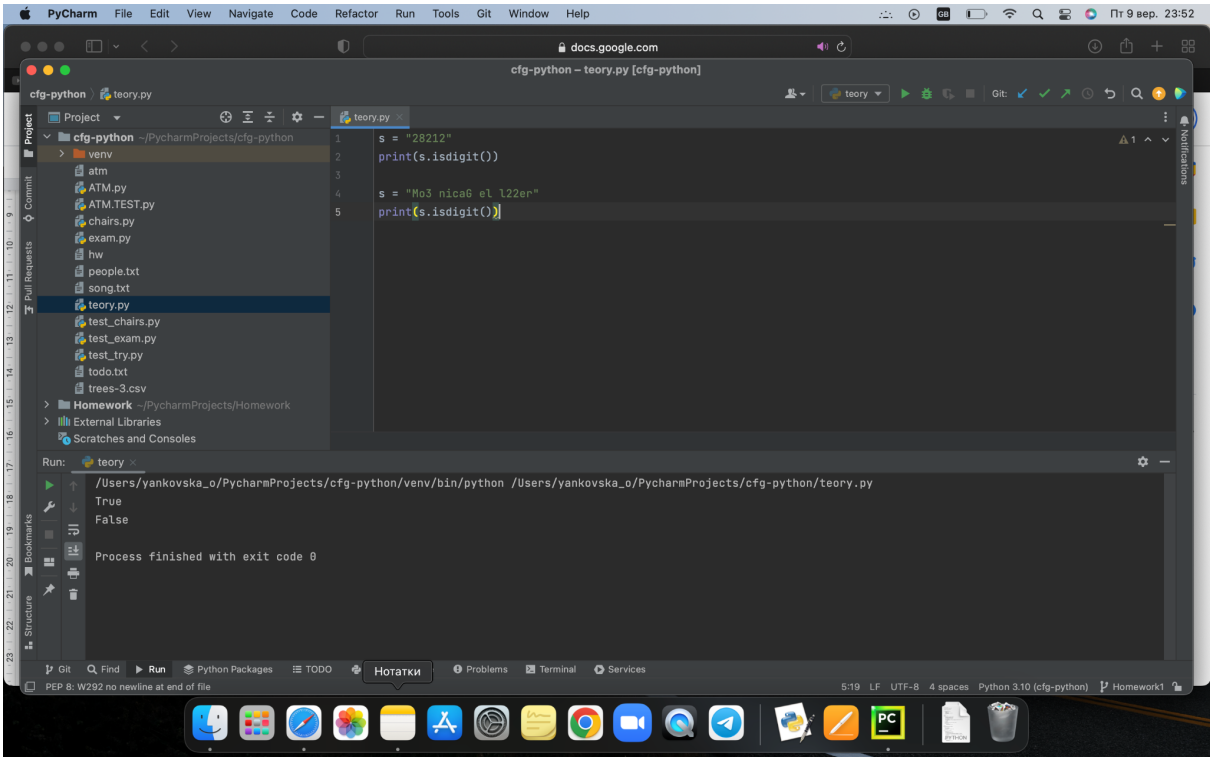


---

isdigit()
The isdigit() method returns True if all characters in a string are digits. If not, it returns False. False if at least one character is not a digit.
The syntax of isdigit() is
string.isdigit()

EXAMPLE:
s = "28212"
print(s.isdigit())

s = "Mo3 nicaG el l22er"
print(s.isdigit())



---

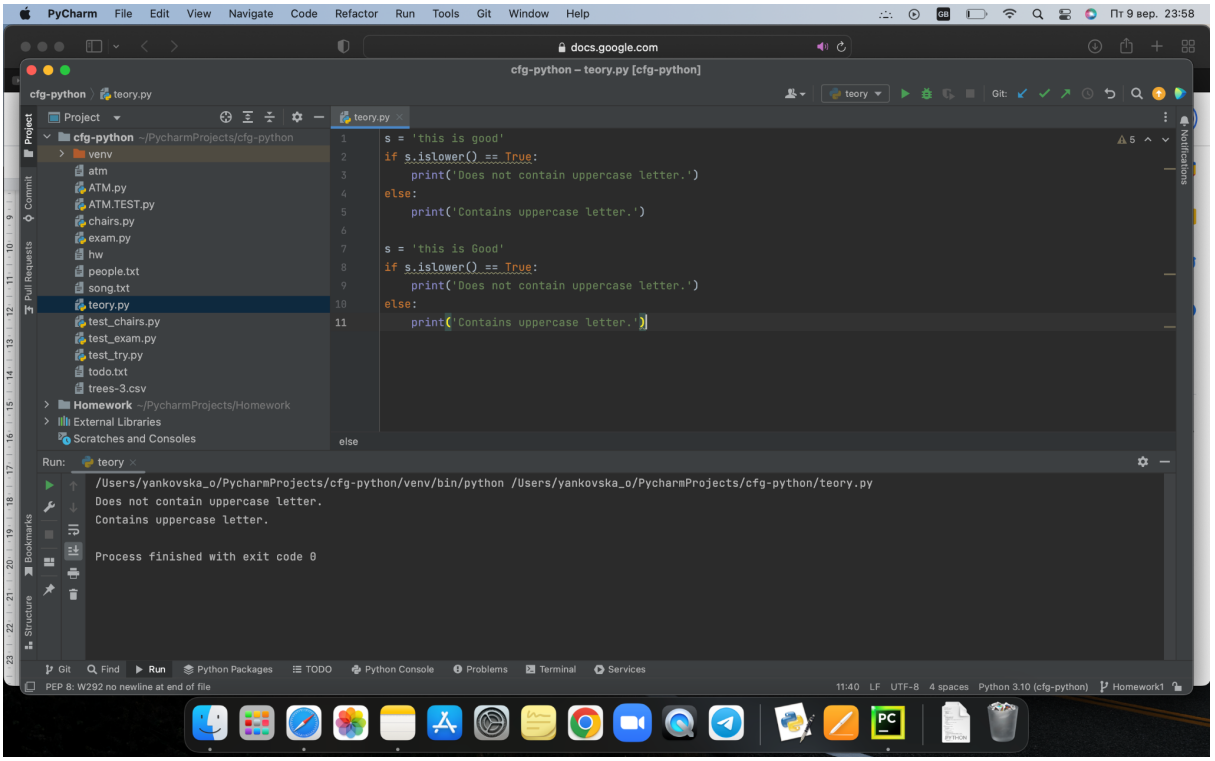
islower()

The islower() method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.

The syntax of islower() is:

```
string.islower()
```

```
s = 'this is good'
if s.islower() == True:
 print('Does not contain uppercase letter.')
else:
 print('Contains uppercase letter.')

s = 'this is Good'
if s.islower() == True:
 print('Does not contain uppercase letter.')
else:
 print('Contains uppercase letter.')
```



---

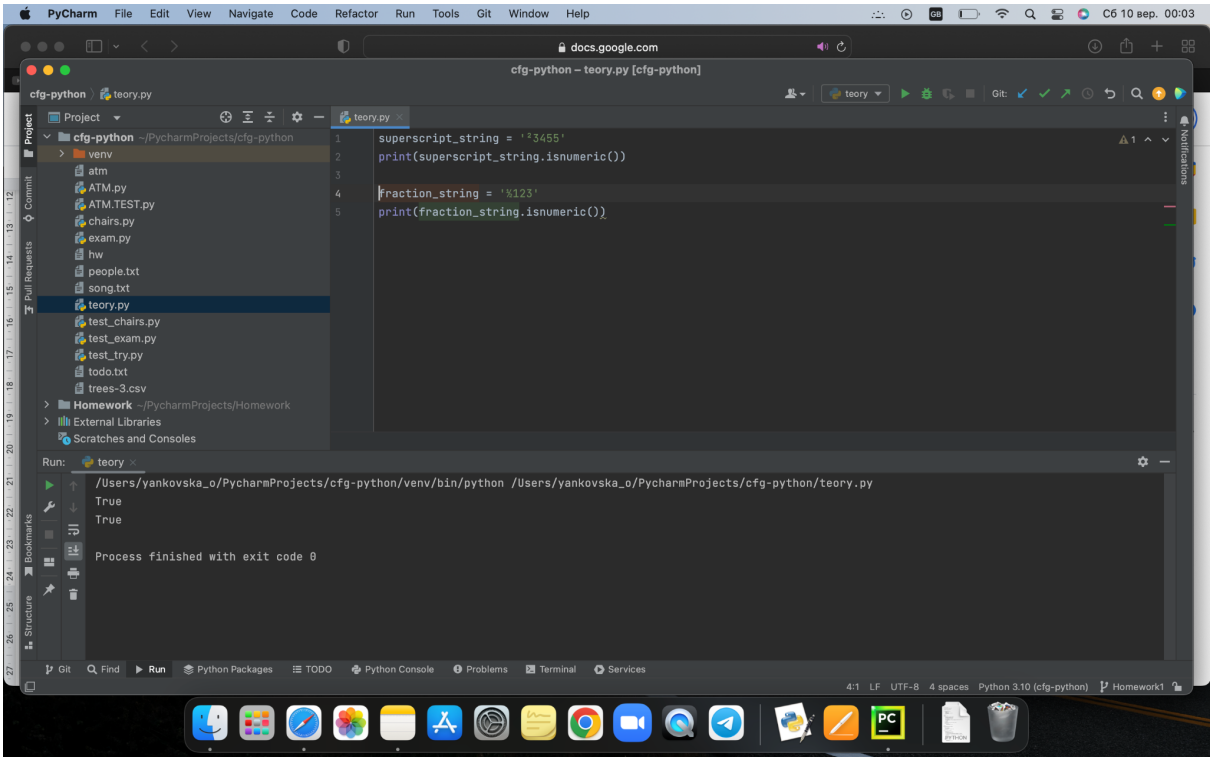
isnumeric()

The isnumeric() method checks if all the characters in the string are numeric.

The syntax of the isnumeric() method is:
string.isnumeric()
Python treats mathematical characters like numbers, subscripts, superscripts, and characters having Unicode numeric value properties (like a fraction, roman numerals, currency numerators) as numeric characters.

EXAMPLE:

```
superscript_string = '²3455'
print(superscript_string.isnumeric())

fraction_string = '½123'
print(fraction_string.isnumeric())
```



---

isspace()

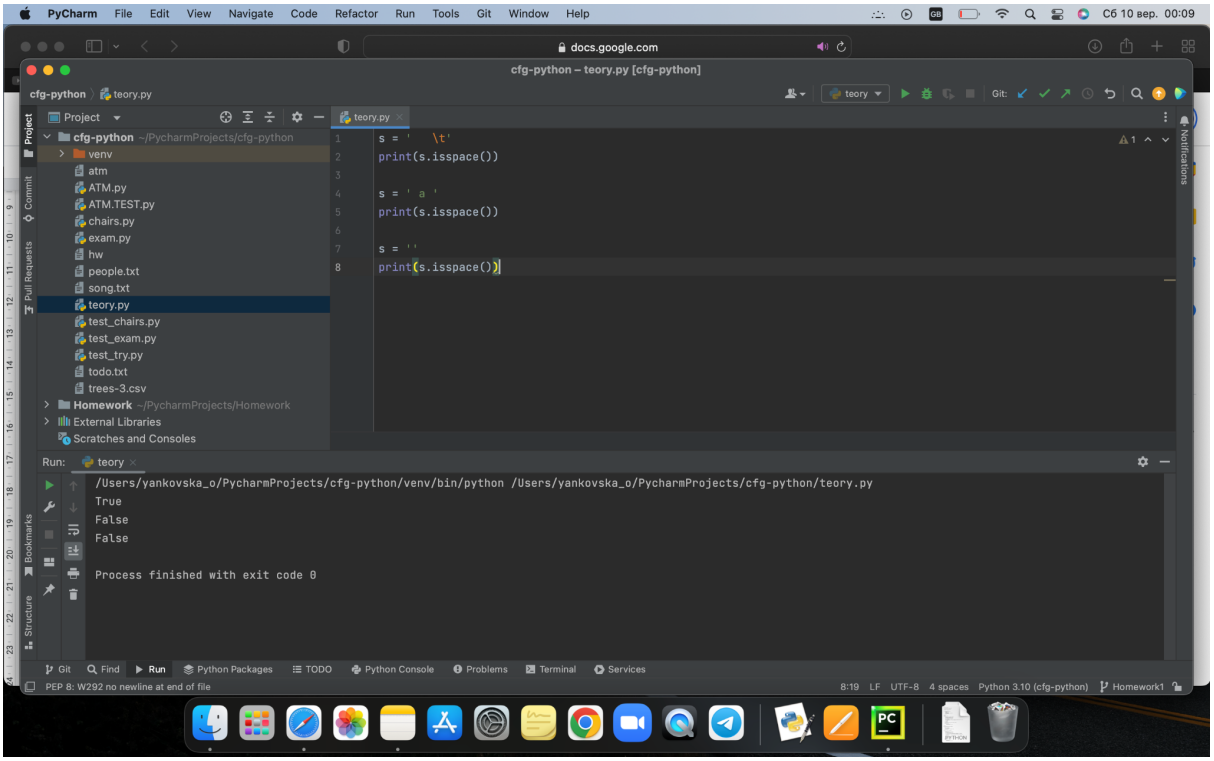
The isspace() method returns True if there are only whitespace characters in the string. False if the string is empty or contains at least one non-printable character

Characters that are used for spacing are called whitespace characters. For example: tabs, spaces, newline, etc.

The syntax of isspace() is:
string.isspace()

EXAMPLE:

```
s = '  \t'
print(s.isspace())

s = 'a'
print(s.isspace())

s = ''
print(s.isspace())
```



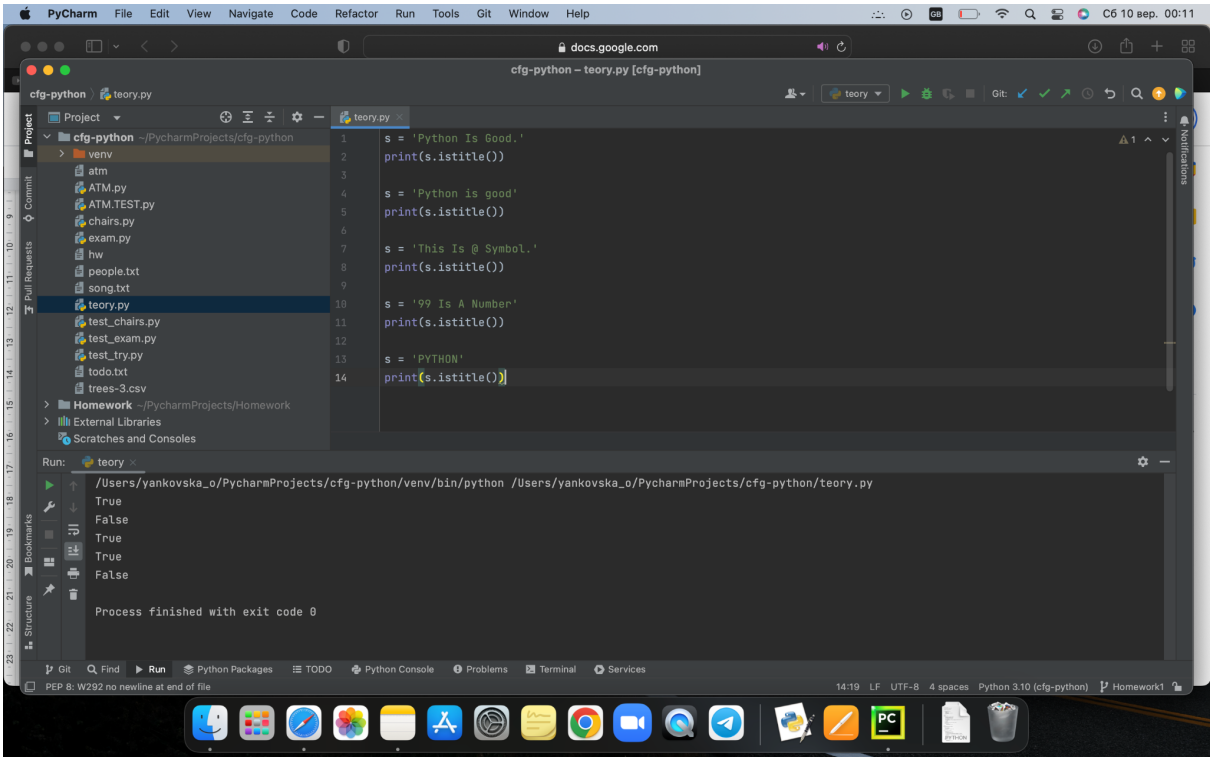
---

istitle()
The istitle() returns True if the string is a title case string. If not, it returns False.

The syntax of istitle() method is:
string.istitle()

EXAMPLE:

```
s = 'Python Is Good.'
print(s.istitle())

s = 'Python is good'
print(s.istitle())

s = 'This Is @ Symbol.'
print(s.istitle())

s = '99 Is A Number'
print(s.istitle())

s = 'PYTHON'
print(s.istitle())
```



---

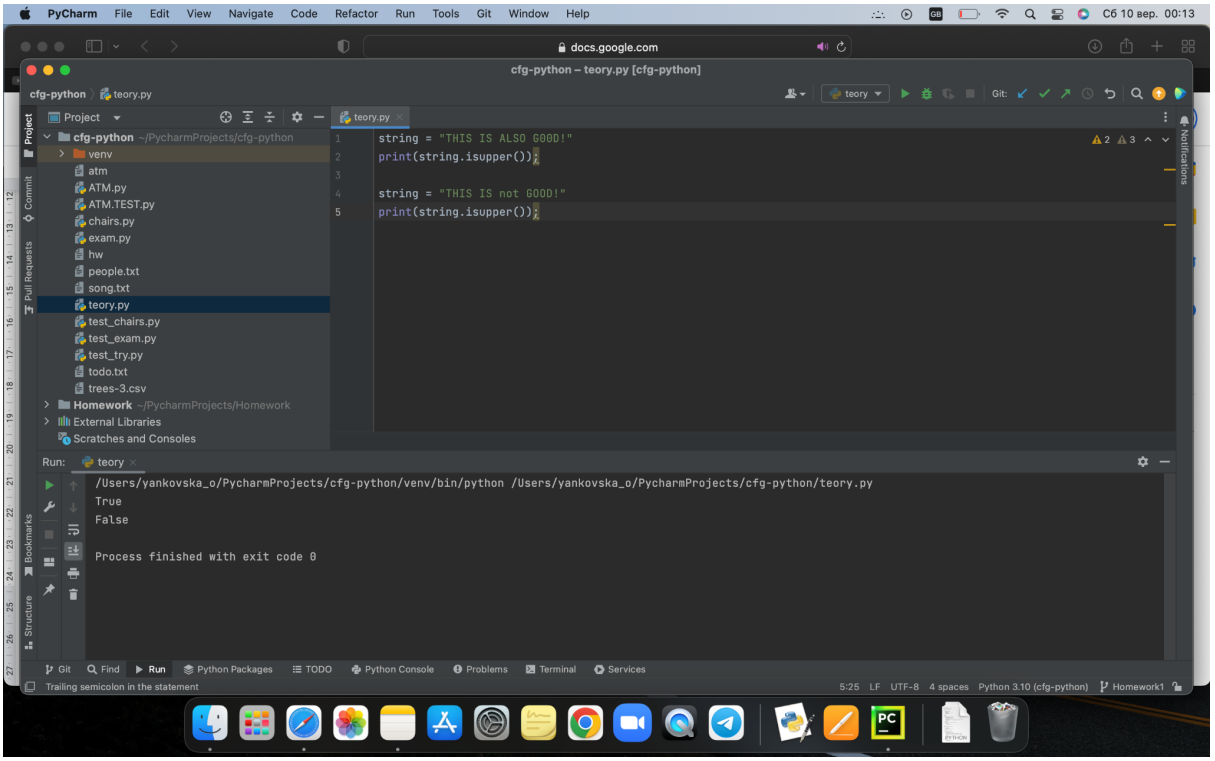
isupper()

The string isupper() method returns whether or not all characters in a string are uppercased or not.

The syntax of isupper() method is:
string.isupper()

EXAMPLE:

```
string = "THIS IS ALSO GOOD!"
print(string.isupper());

string = "THIS IS not GOOD!"
print(string.isupper());
```



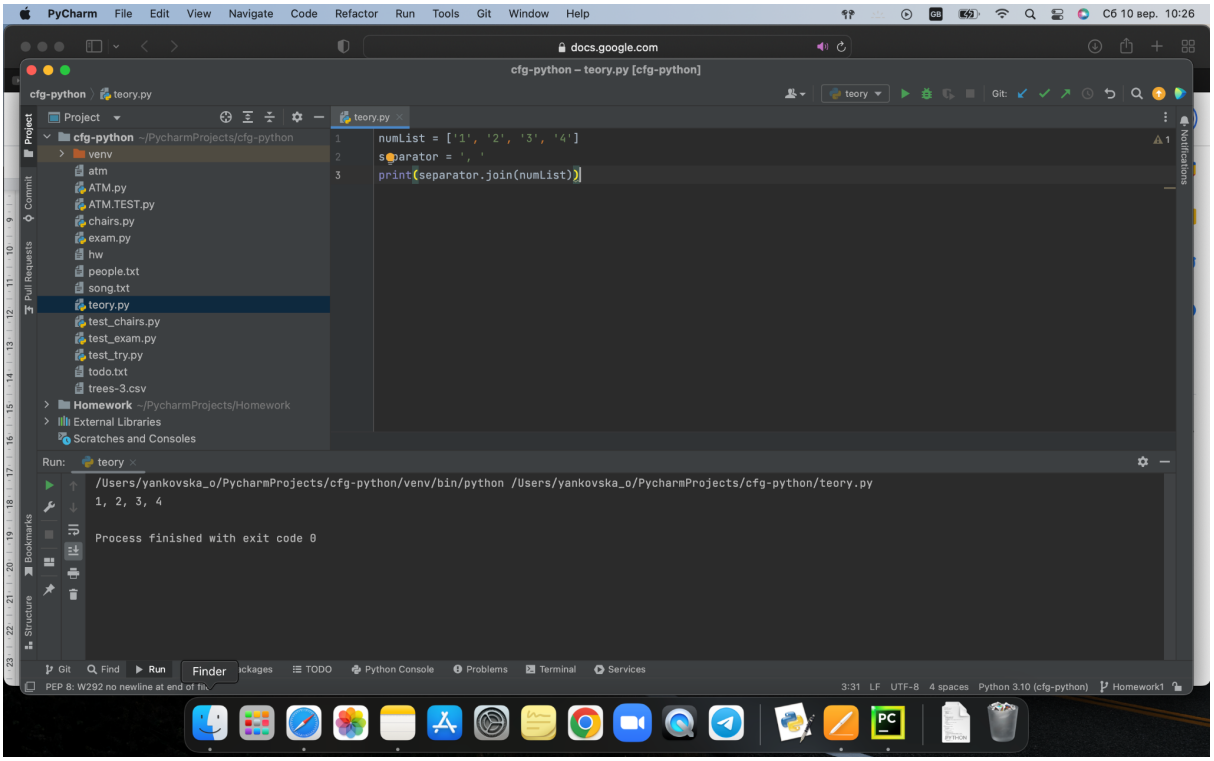
---

join()

The string join() method returns a string by joining all the elements of an iterable (list, string, tuple), separated by the given separator.

Example

```
numList = ['1', '2', '3', '4']
separator = ', '
print(separator.join(numList))
```

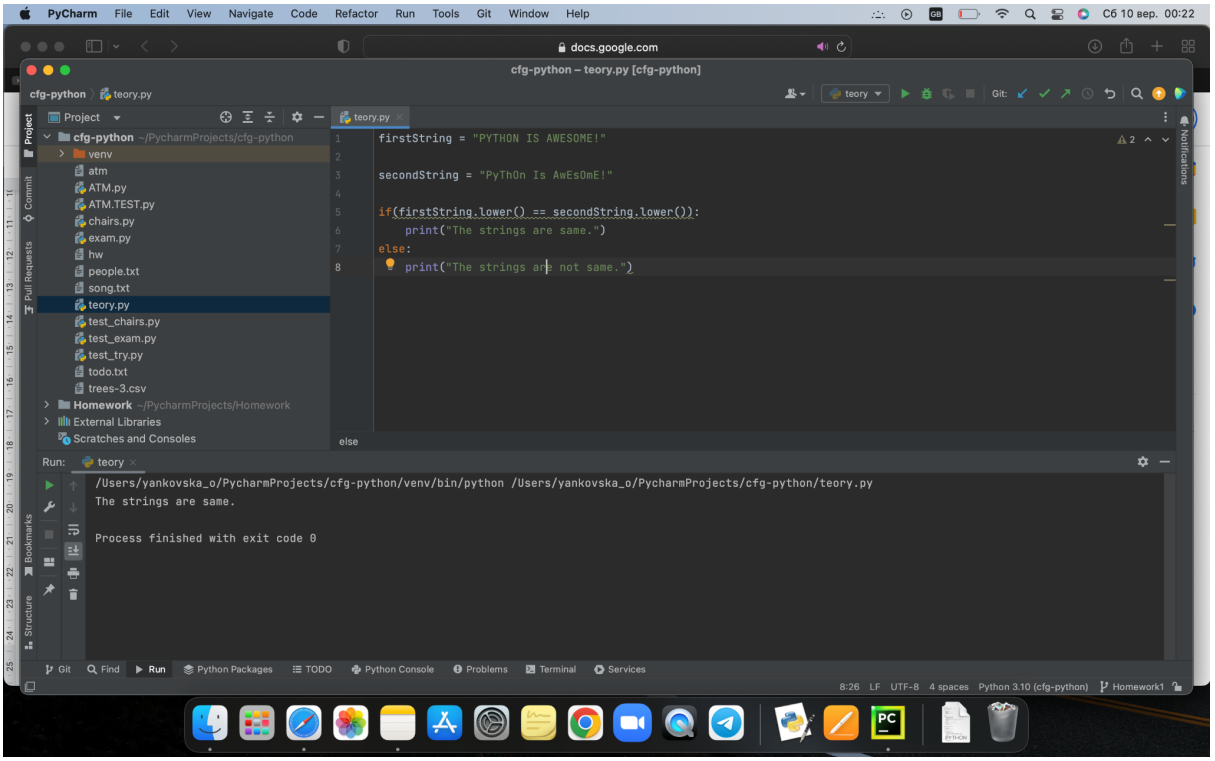


---

lower()
The lower() method converts all uppercase characters in a string into lowercase characters and returns it.
The syntax of lower() method is:
string.lower()

EXAMPLE:

```
firstString = "PYTHON IS AWESOME!"

secondString = "PyThOn Is AwEsOmE!"

if(firstString.lower() == secondString.lower()):
   print("The strings are same.")
else:
   print("The strings are not same.")
```



---

lstrip()

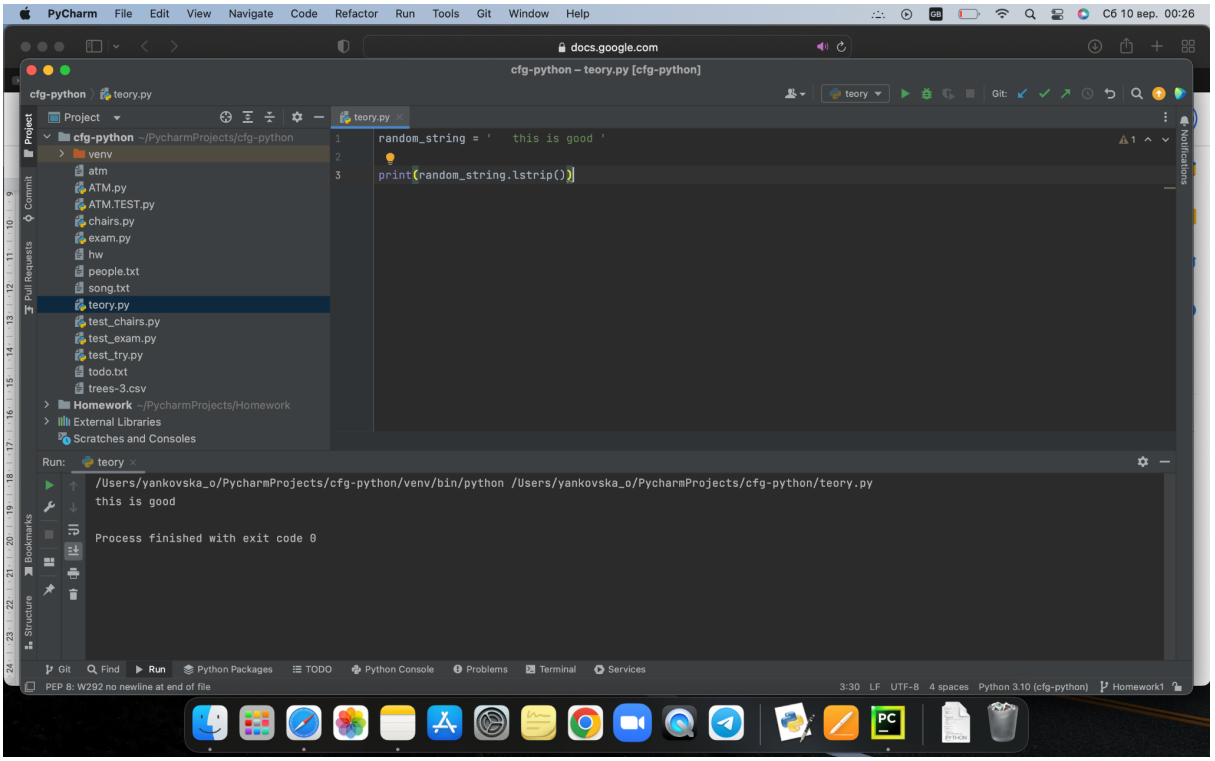
The lstrip() method returns a copy of the string with leading characters removed (based on the string argument passed).

The lstrip() removes characters from the left based on the argument (a string specifying the set of characters to be removed).

The syntax of lstrip() is:
string.lstrip([chars])

lstrip() Parameters
chars (optional) - a string specifying the set of characters to be removed.
If chars argument is not provided, all leading whitespaces are removed from the string.
EXAMPLE:

```
random_string = '   this is good '
print(random_string.lstrip())
```

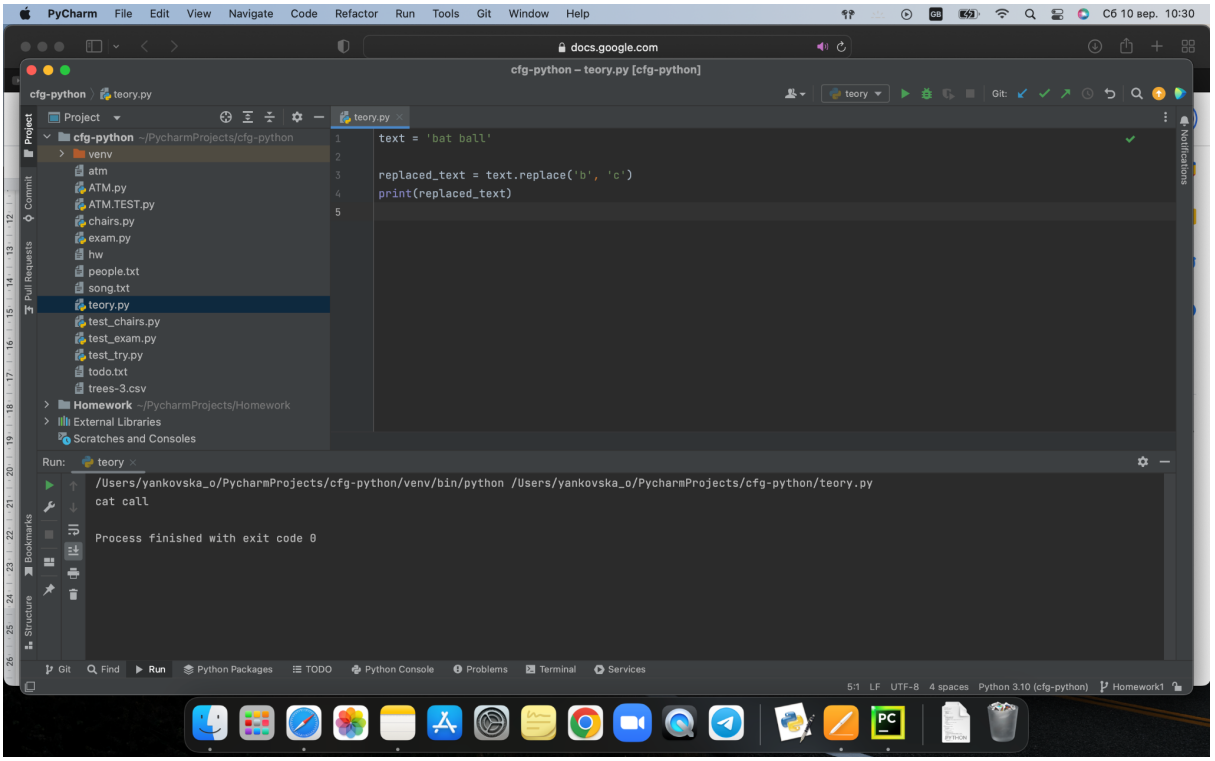


---

replace()
The replace() method replaces each matching occurrence of the old character/text in the string with the new character/text.
It's syntax is:
str.replace(old, new [, count])

The replace() method can take maximum of 3 parameters:
old - old substring you want to replace
new - new substring which will replace the old substring
count (optional) - the number of times you want to replace the old substring with the new substring

```
text = 'bat ball'

replaced_text = text.replace('b', 'c')
print(replaced_text)
```



---

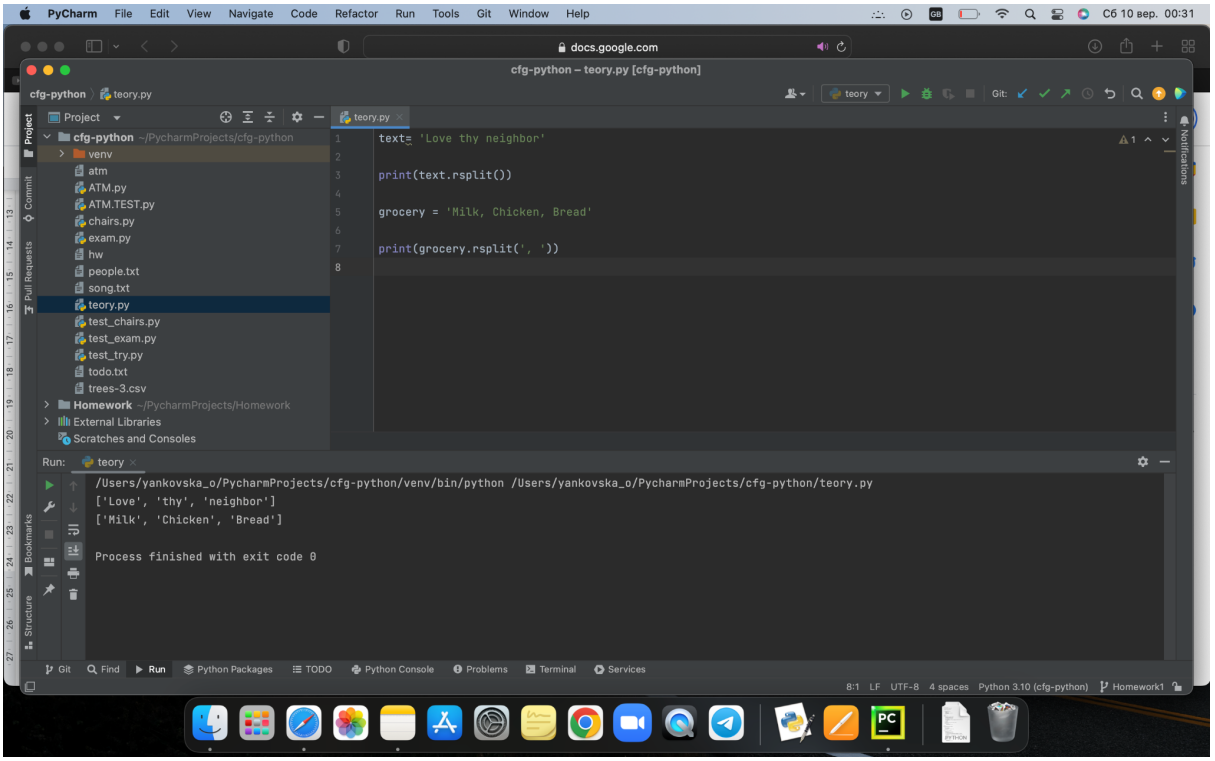
rsplit()
The rsplit() method splits string from the right at the specified separator and returns a list of strings.

The syntax of rsplit() is:
str.rsplit([separator [, maxsplit]])

rsplit() method takes maximum of 2 parameters:
separator (optional)- The is a delimiter. rsplit() method splits string starting from the right at the specified separator.
If the separator is not specified, any whitespace (space, newline etc.) string is a separator.
maxsplit (optional) - The maxsplit defines the maximum number of splits.
The default value of maxsplit is -1, meaning, no limit on the number of splits.

EXAMPLE:
text= 'Love thy neighbor'
print(text.rsplit())

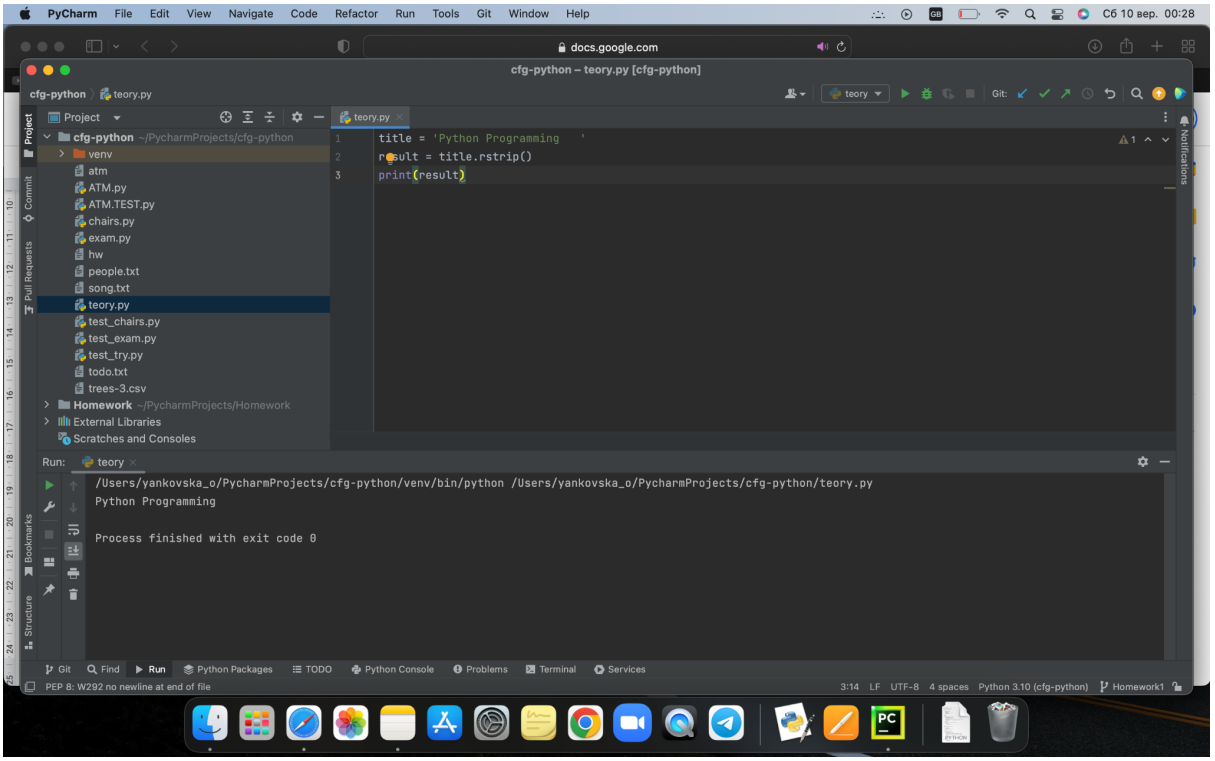
grocery = 'Milk, Chicken, Bread'
print(grocery.rsplit(', '))



---

rstrip()
The rstrip() method returns a copy of the string with trailing characters removed (based on the string argument passed).
EXAMPLE:

```
title = 'Python Programming  '
result = title.rstrip()
print(result)
```



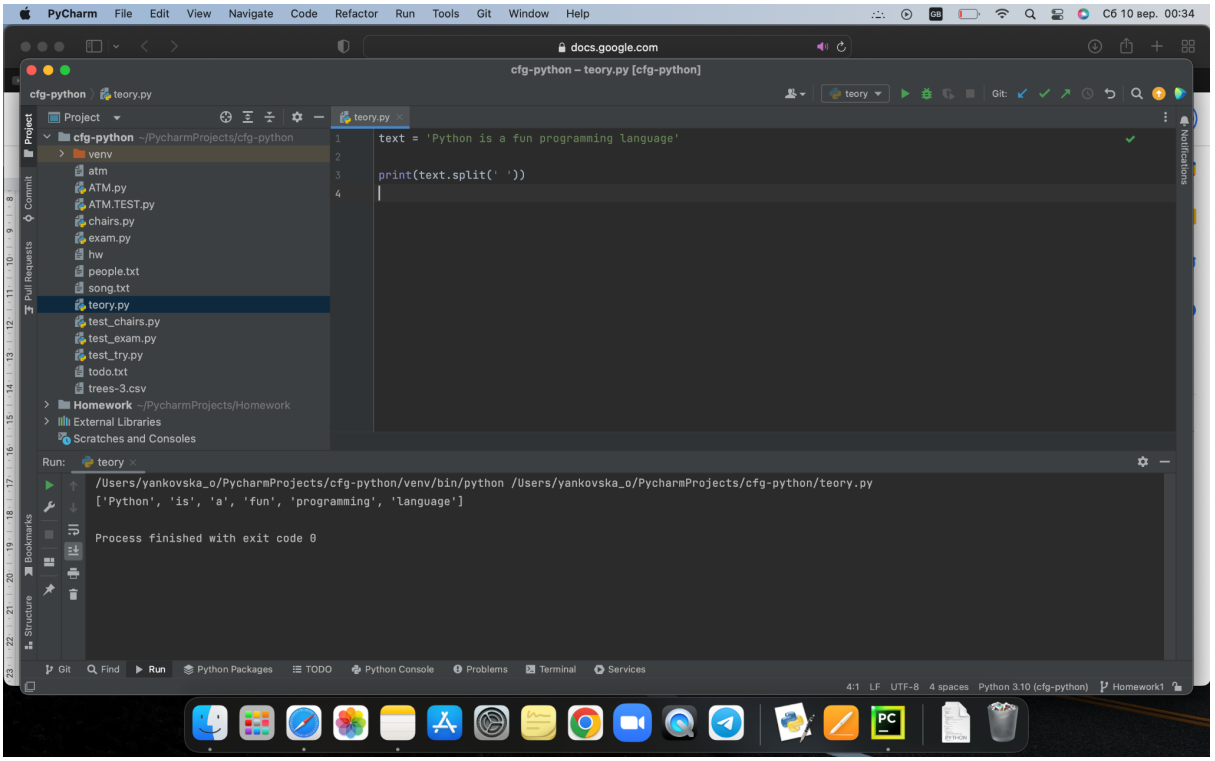
---

split()

The split() method breaks up a string at the specified separator and returns a list of strings.
The syntax of split() is:
str.split(separator, maxsplit)
The split() method takes a maximum of 2 parameters:
separator (optional)- Delimiter at which splits occur. If not provided, the string is splitted at whitespaces.
maxsplit (optional) - Maximum number of splits. If not provided, there is no limit on the number of splits.

EXAMPLE:

```
text = 'Python is a fun programming language'
print(text.split(' '))
```

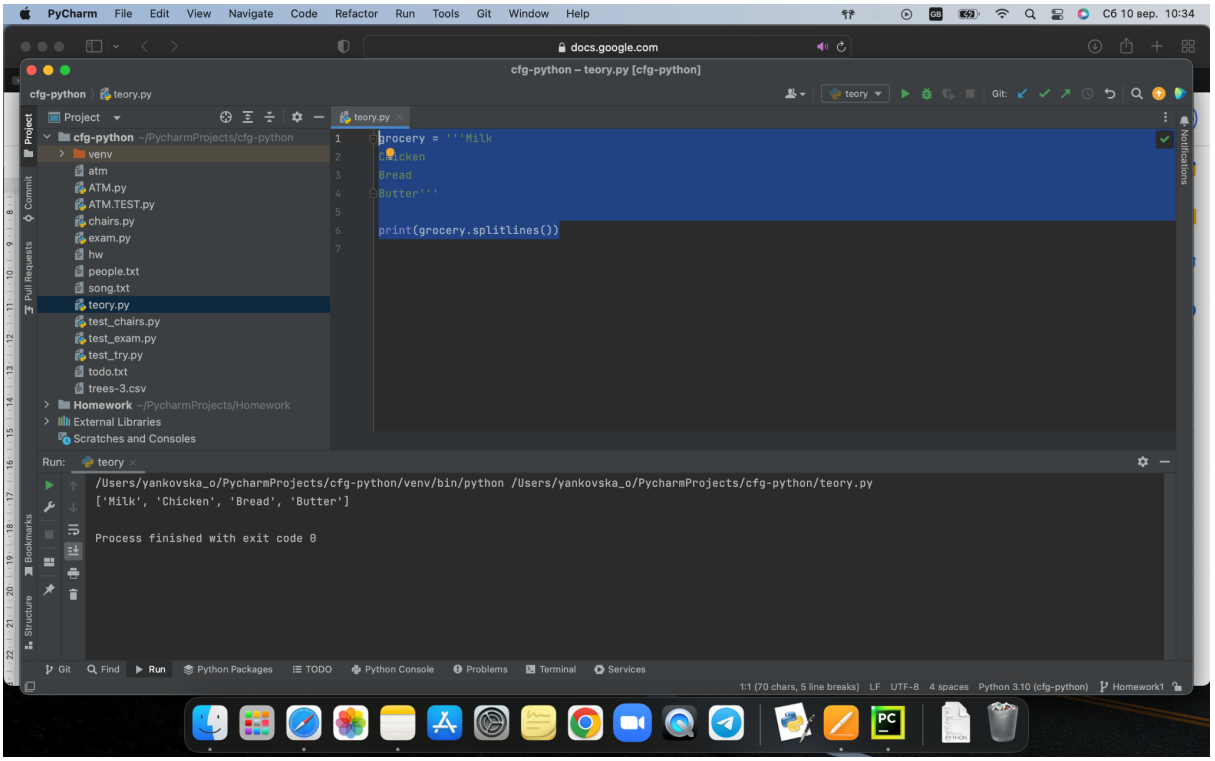


---

splitlines()
The splitlines() method splits the string at line breaks and returns a list..
The syntax of the splitlines() method is:
string.splitlines([keepends])

The splitlines() method can take a single parameter:
keepends(optional) - it determines whether line breaks are included in the resulting list or not. It's value can be True or any number.

```
grocery = '''Milk
Chicken
Bread
Butter'''

print(grocery.splitlines())
```



---

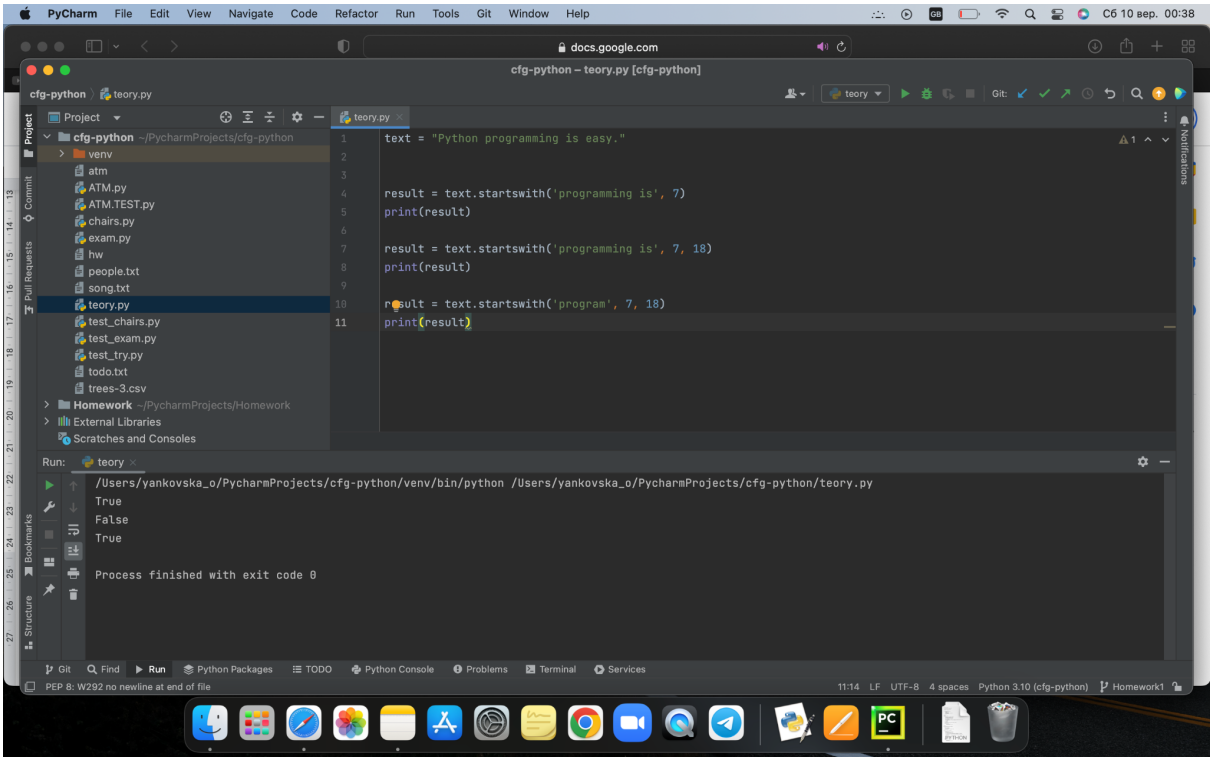
startswith()
The startswith() method returns True if a string starts with the specified prefix(string). If not, it returns False.

startswith() method takes a maximum of three parameters:

prefix - String or tuple of strings to be checked
start (optional) - Beginning position where prefix is to be checked within the string.
end (optional) - Ending position where prefix is to be checked within the string.

EXAMPLE:

```
text = "Python programming is easy."

result = text.startswith('programming is', 7)
print(result)

result = text.startswith('programming is', 7, 18)
print(result)

result = text.startswith('program', 7, 18)
print(result)
```



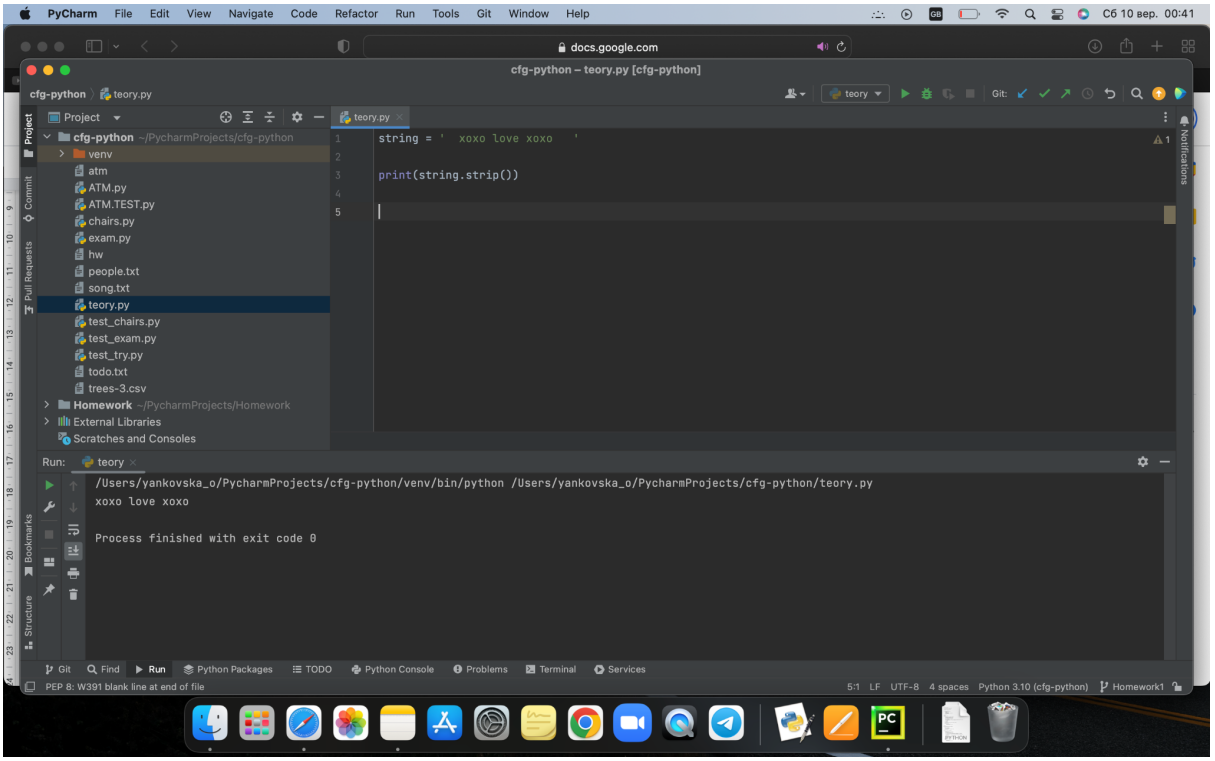
---

strip()

The strip() method returns a copy of the string by removing both the leading and the trailing characters (based on the string argument passed).

chars (optional) - a string specifying the set of characters to be removed from the left and right part of the string.
The strip() method removes characters from both left and right based on the argument (a string specifying the set of characters to be removed).

Note: If the chars argument is not provided, all leading and trailing whitespaces are removed from the string.
string = ' xoxo love xoxo '
print(string.strip())

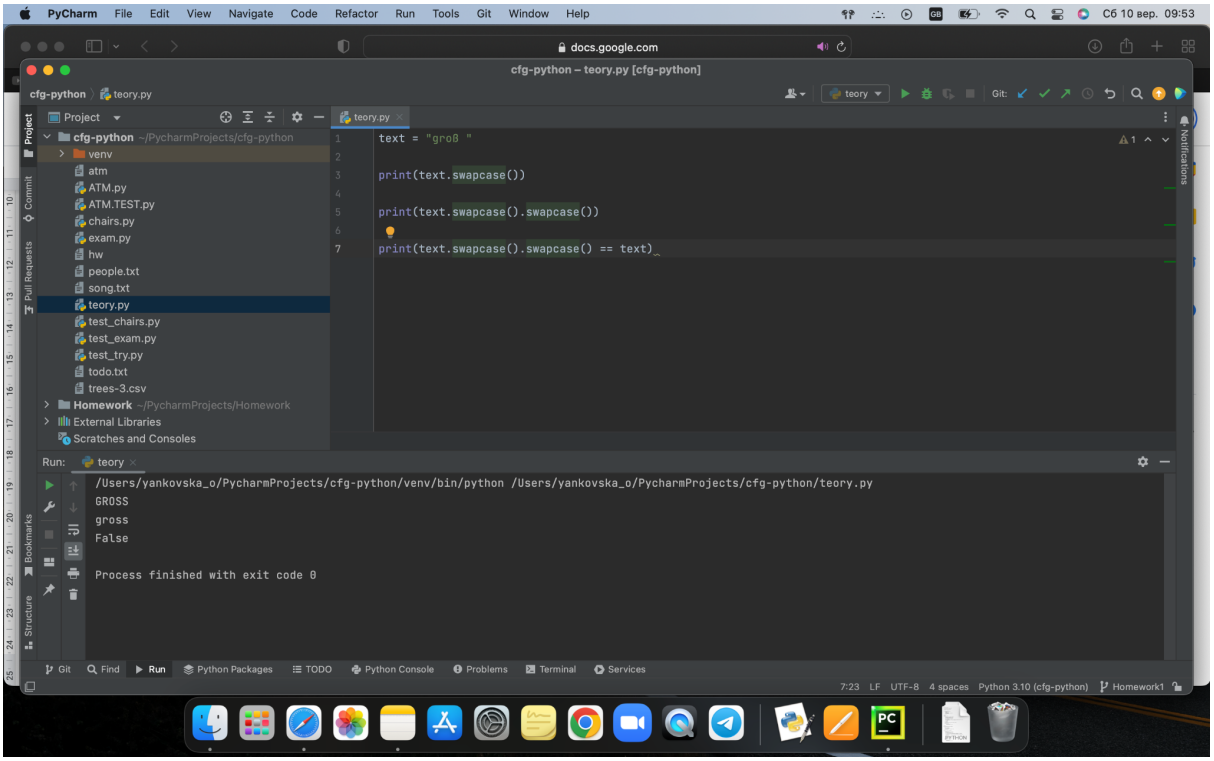


---

swapcase()

The swapcase() method returns the string by converting all the characters to their opposite letter case(uppercase to lowercase and vice versa).
The syntax of the swapcase() method is:
string.swapcase()
EXAMPLE:
text = "groß "

```
print(text.swapcase())
print(text.swapcase().swapcase())
print(text.swapcase().swapcase() == text)
```



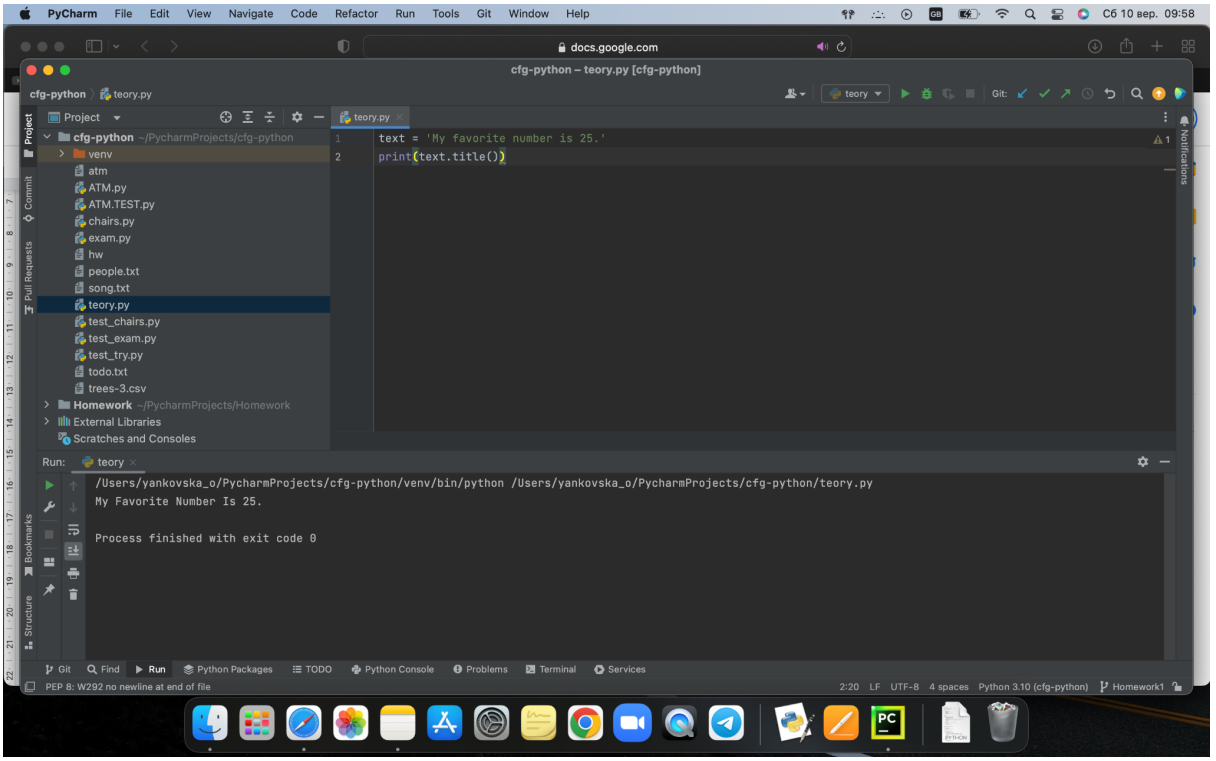
---

title()
The title() method returns a string with first letter of each word capitalized; a title cased string.

The syntax of title() is:
str.title()

EXAMPLE:

text = 'My favorite number is 25.'
print(text.title())



---

upper()

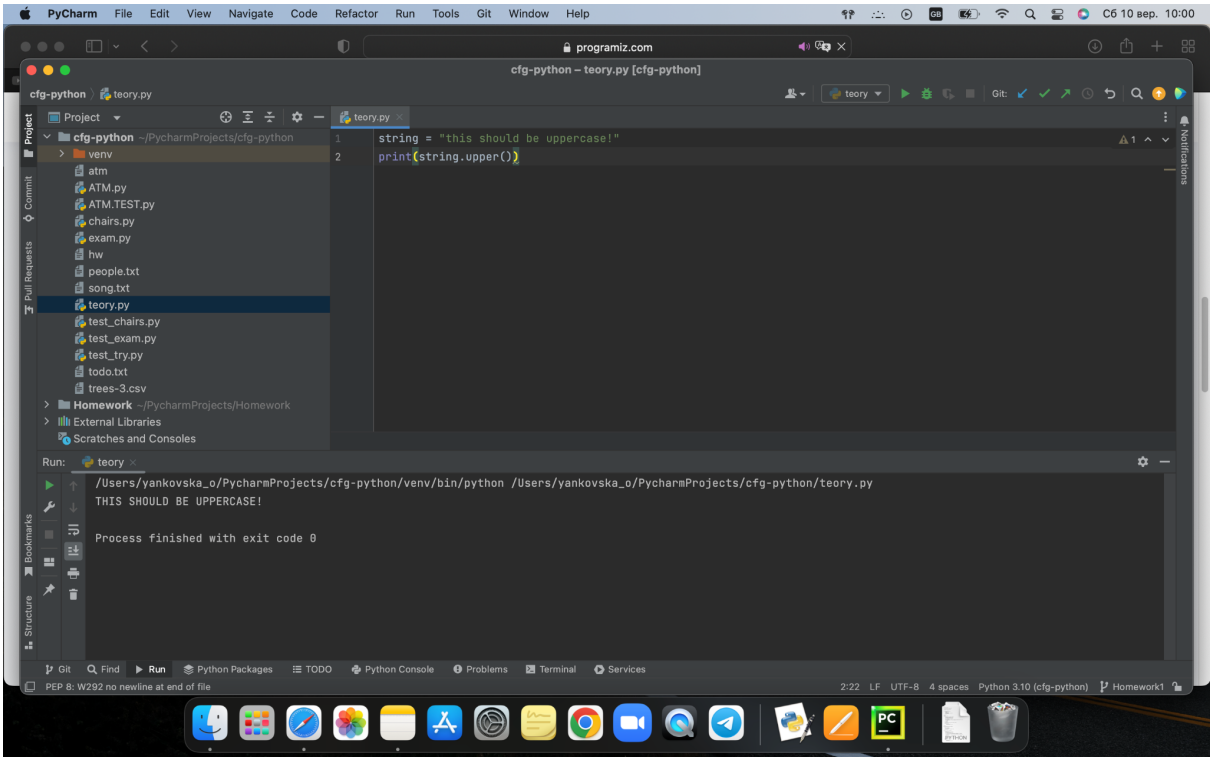
The upper() method converts all lowercase characters in a string into uppercase characters and returns it.

The syntax of upper() method is:

string.upper()

EXAMPLE:

```
string = "this should be uppercase!"
print(string.upper())
```



---

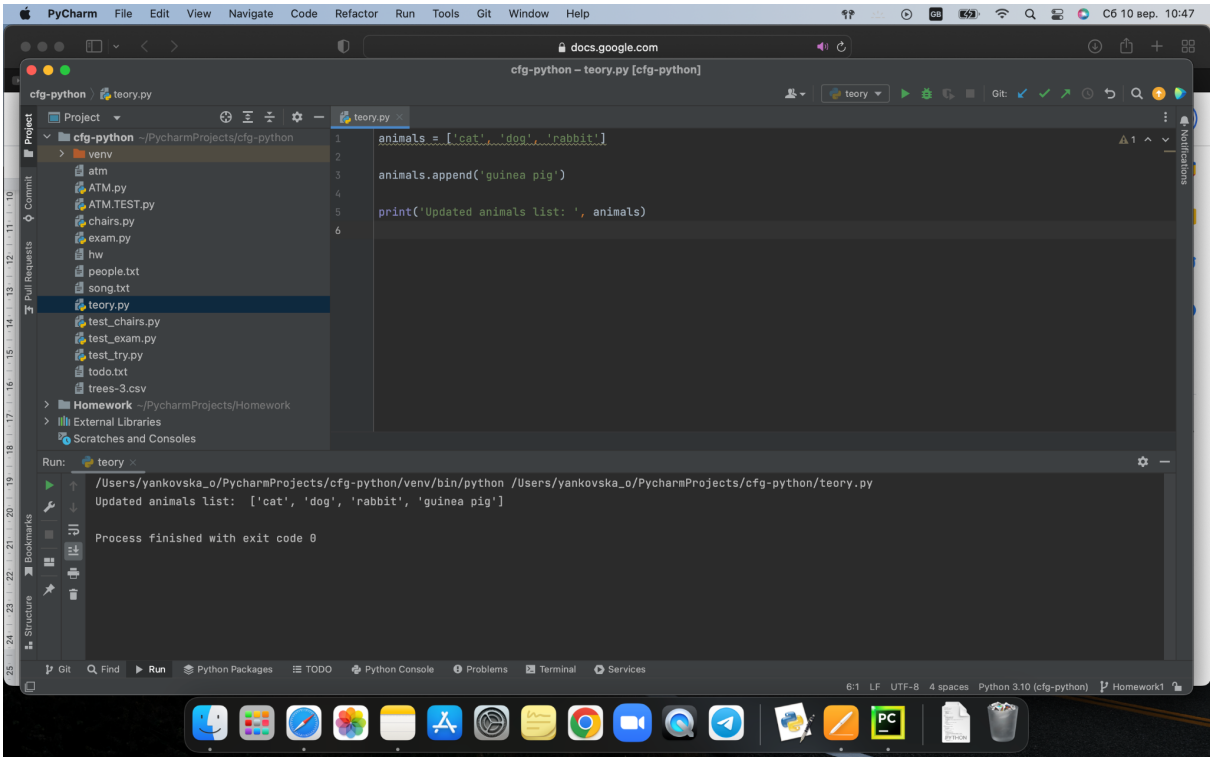
3. Python list methods:
describe each method and provide an example

append()
The append() method adds an item to the end of the list.
The syntax of the append() method is:
list.append(item)

The method takes a single argument
item - an item (number, string, list etc.) to be added at the end of the list

EXAMPLE:
animals = ['cat', 'dog', 'rabbit']
animals.append('guinea pig')

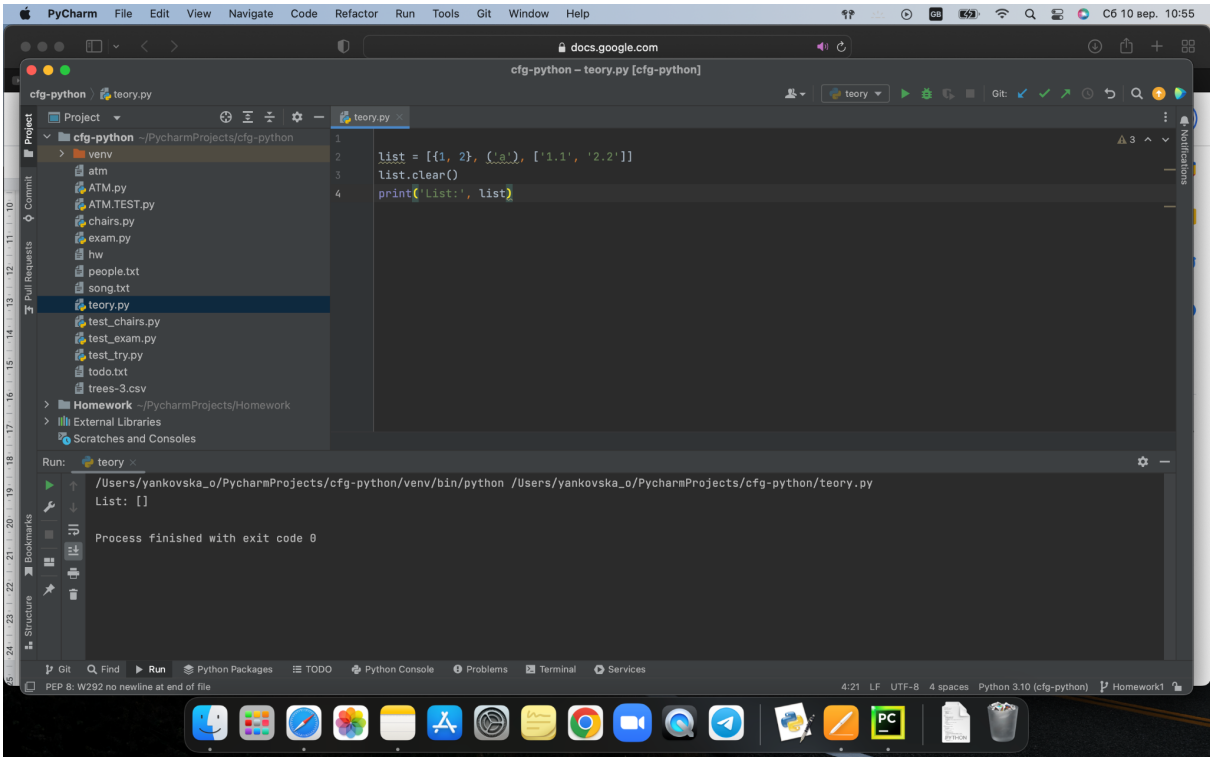
print('Updated animals list: ', animals)



---

clear()
The clear() method removes all items from the list.
The syntax of clear() method is:
list.clear()

EXAMPLE:
list = [{1, 2}, ('a'), ['1.1', '2.2']]
list.clear()
print('List:', list)



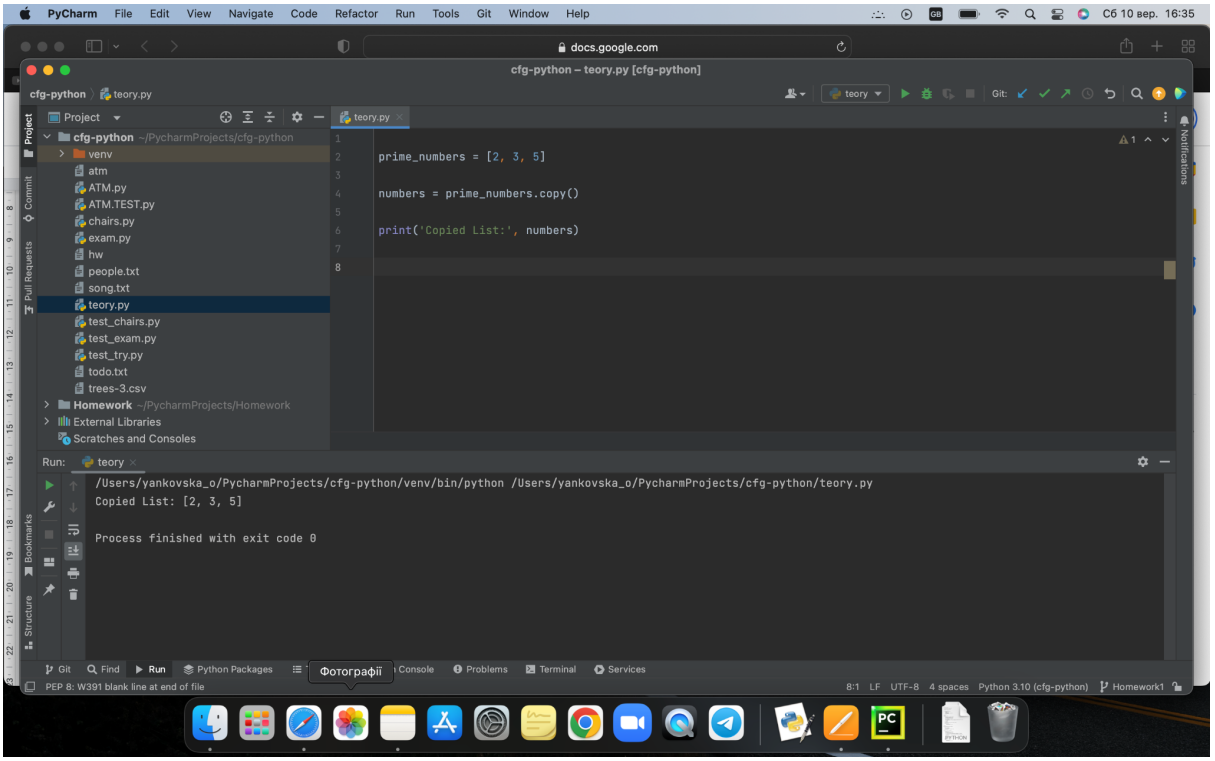
---

copy()
The copy() method returns a shallow copy of the list.

The syntax of the copy() method is:
new_list = list.copy()

EXAMPLE:
prime_numbers = [2, 3, 5]
numbers = prime_numbers.copy()

print('Copied List:', numbers)

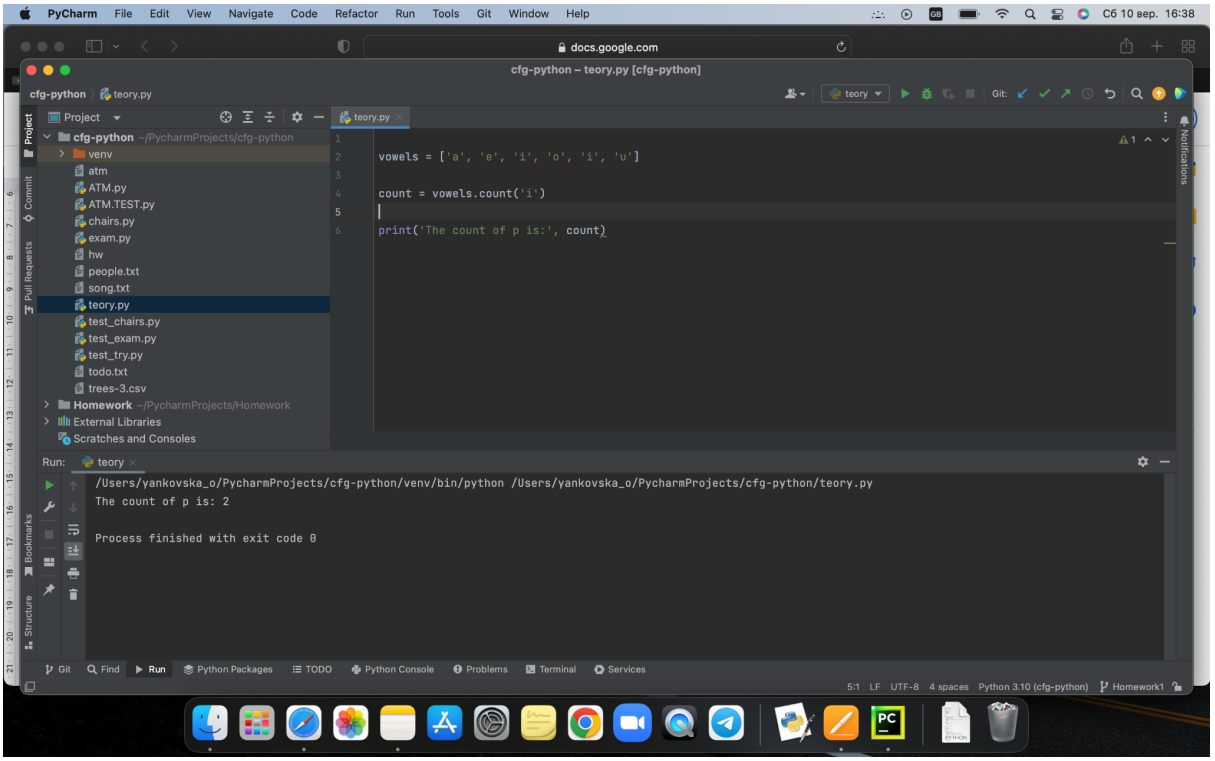


---

count()
The count() method returns the number of times the specified element appears in the list.
The syntax of the count() method is:
list.count(element)
element - the element to be counted

EXAMPLE:
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
count = vowels.count('i')

print('The count of p is:', count)

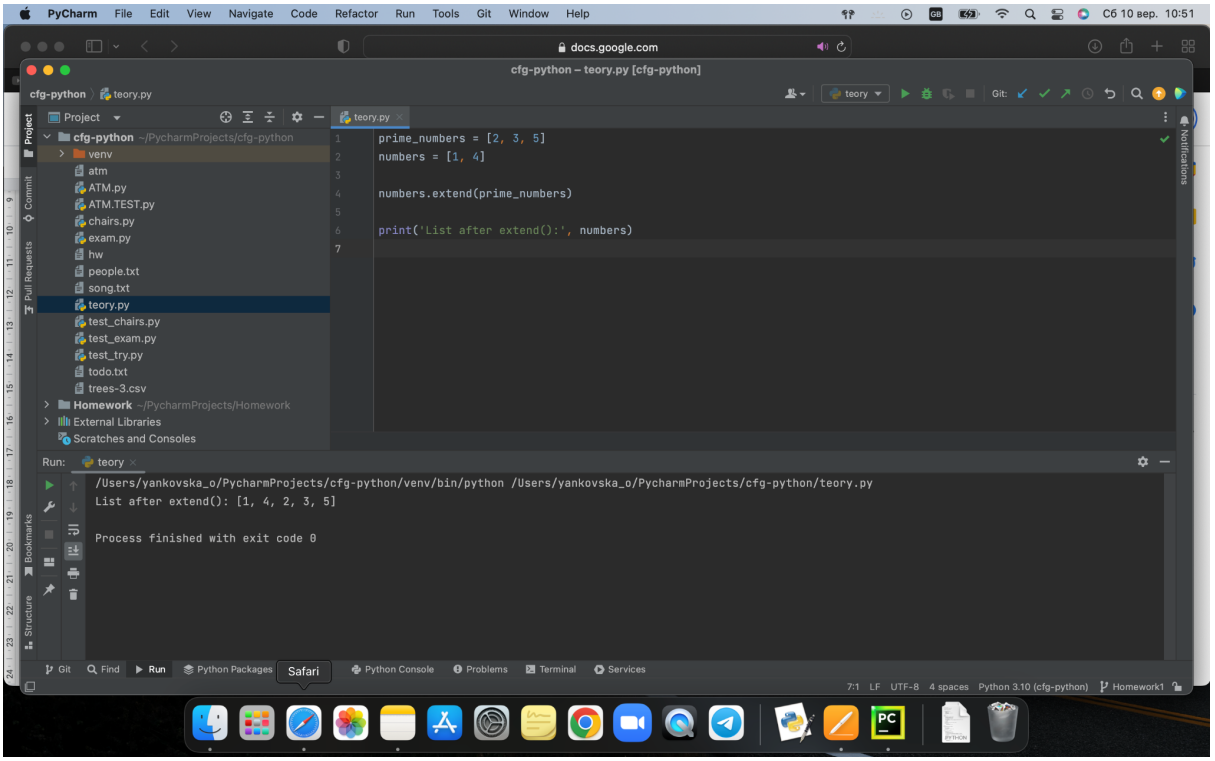


---

extend()
The extend() method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.
The syntax of the extend() method is:
list1.extend(iterable)
the extend() method takes an iterable such as list, tuple, string etc.

EXAMPLE:

```
prime_numbers = [2, 3, 5]
numbers = [1, 4]

numbers.extend(prime_numbers)

print('List after extend():', numbers)
```



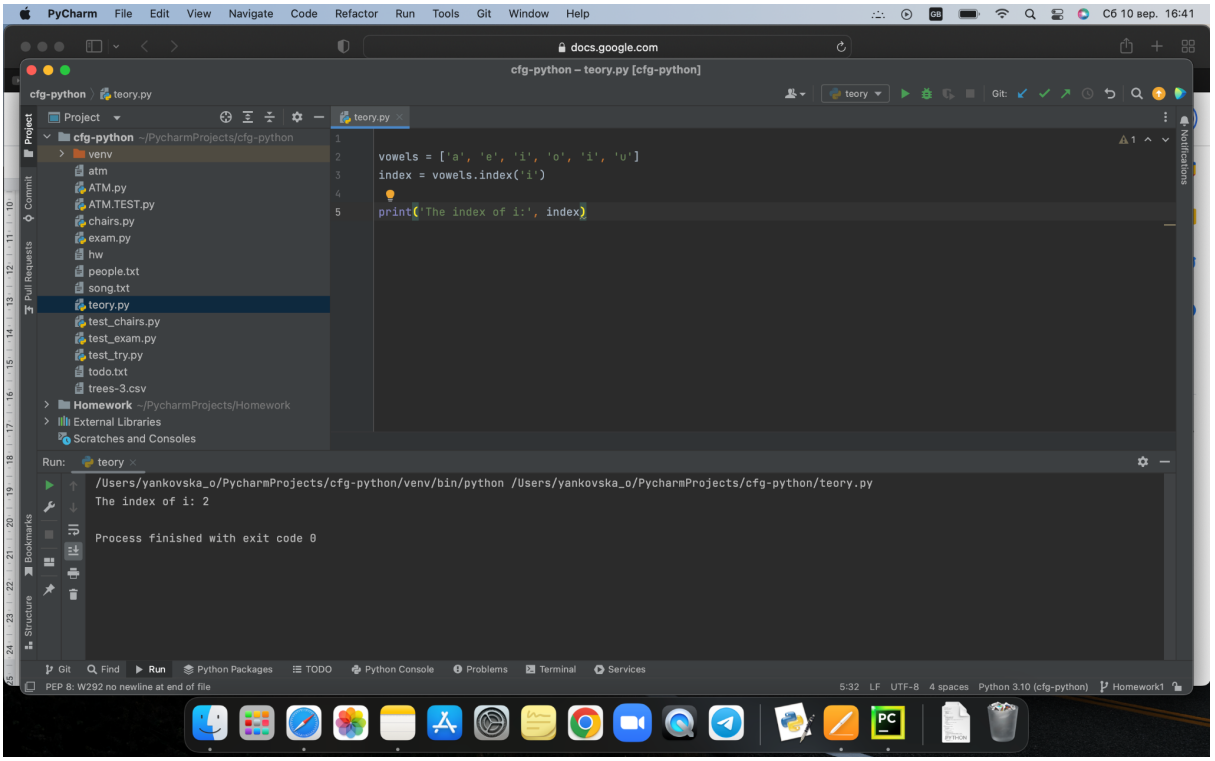
---

index()
The index() method returns the index of the specified element in the list.
The syntax of the list index() method is:
list.index(element, start, end)

The list index() method can take a maximum of three arguments:
element - the element to be searched
start (optional) - start searching from this index
end (optional) - search the element up to this index

If the element is not found, a ValueError exception is raised.
The index() method only returns the first occurrence of the matching element.

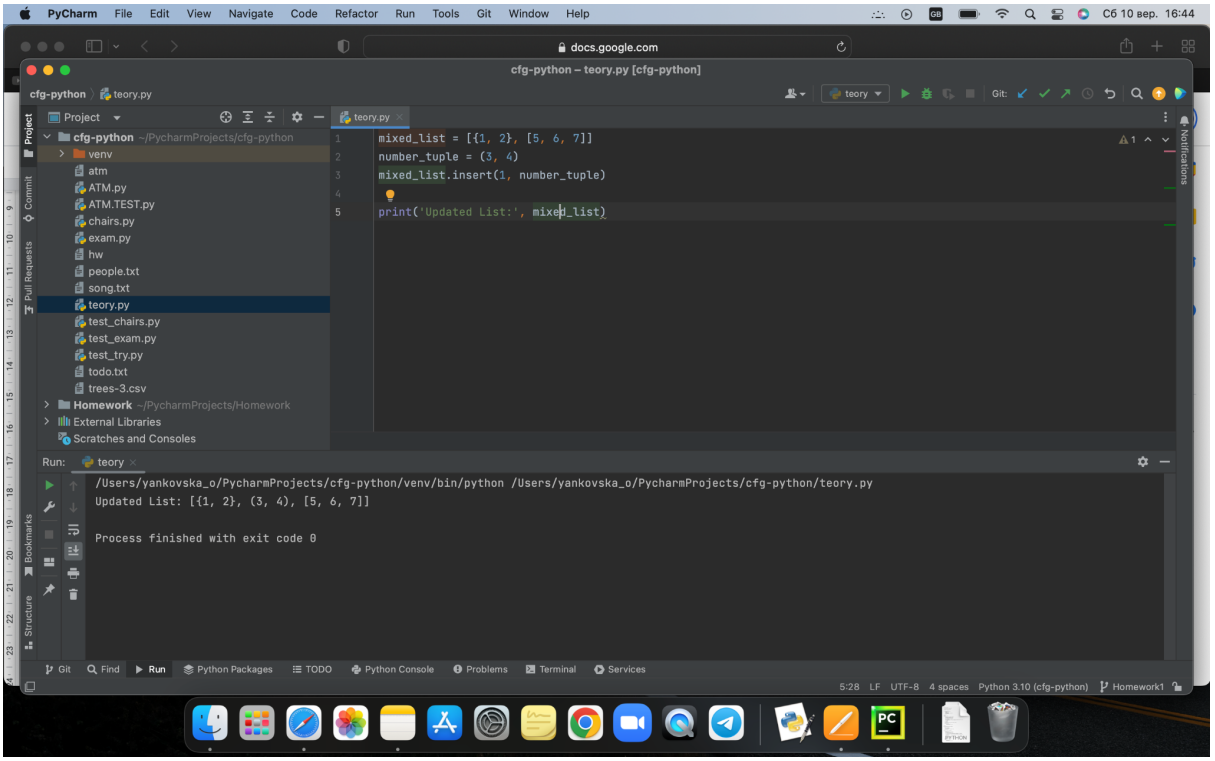
```
vowels = ['a', 'e', 'i', 'o', 'i', 'u']
index = vowels.index('i')

print('The index of i:', index)
```



---

insert()
The insert() method inserts an element to the list at the specified index.
The syntax of the insert() method is
list.insert(i, elem)
The insert() method takes two parameters:
index - the index where the element needs to be inserted
element - this is the element to be inserted in the list

```
mixed_list = [{1, 2}, [5, 6, 7]]
number_tuple = (3, 4)
mixed_list.insert(1, number_tuple)

print('Updated List:', mixed_list)
```



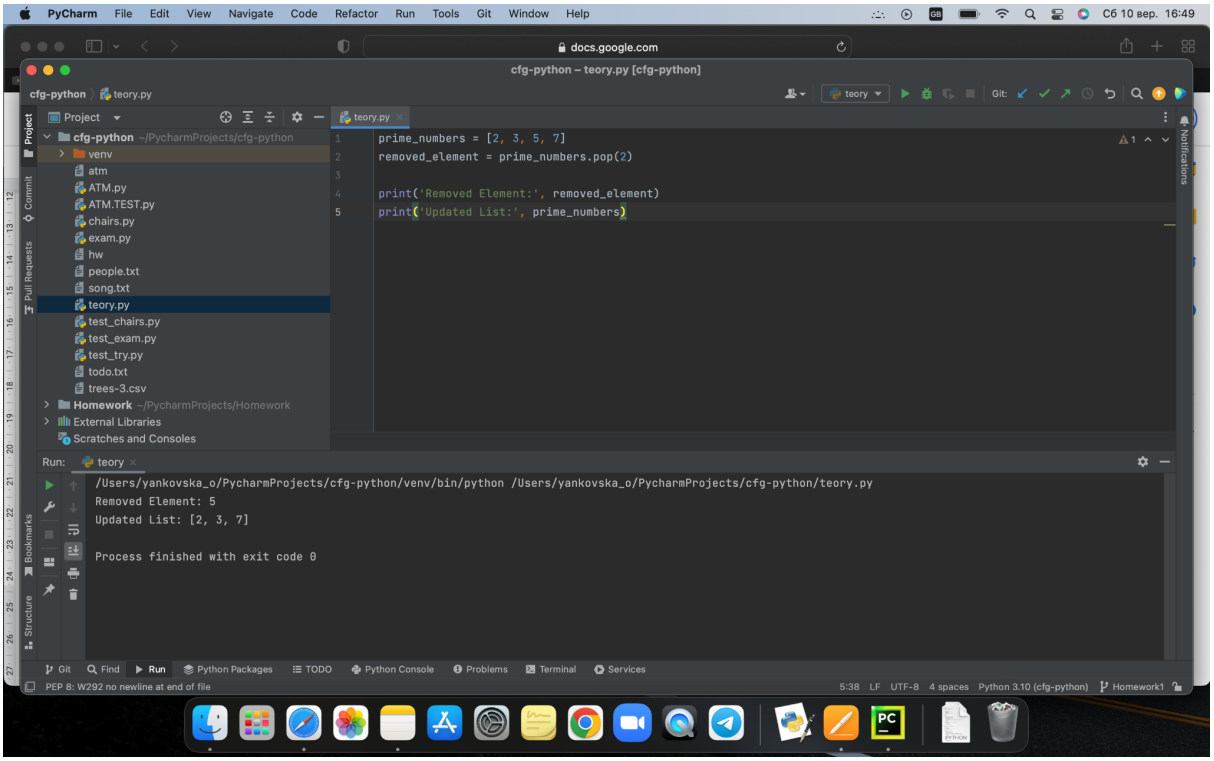
---

pop()

The pop() method removes the item at the given index from the list and returns the removed item.
The syntax of the pop() method is:
list.pop(index)

The pop() method takes a single argument (index).
The argument passed to the method is optional. If not passed, the default index -1 is passed as an argument (index of the last item).
If the index passed to the method is not in range, it throws IndexError: pop index out of range exception.

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

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

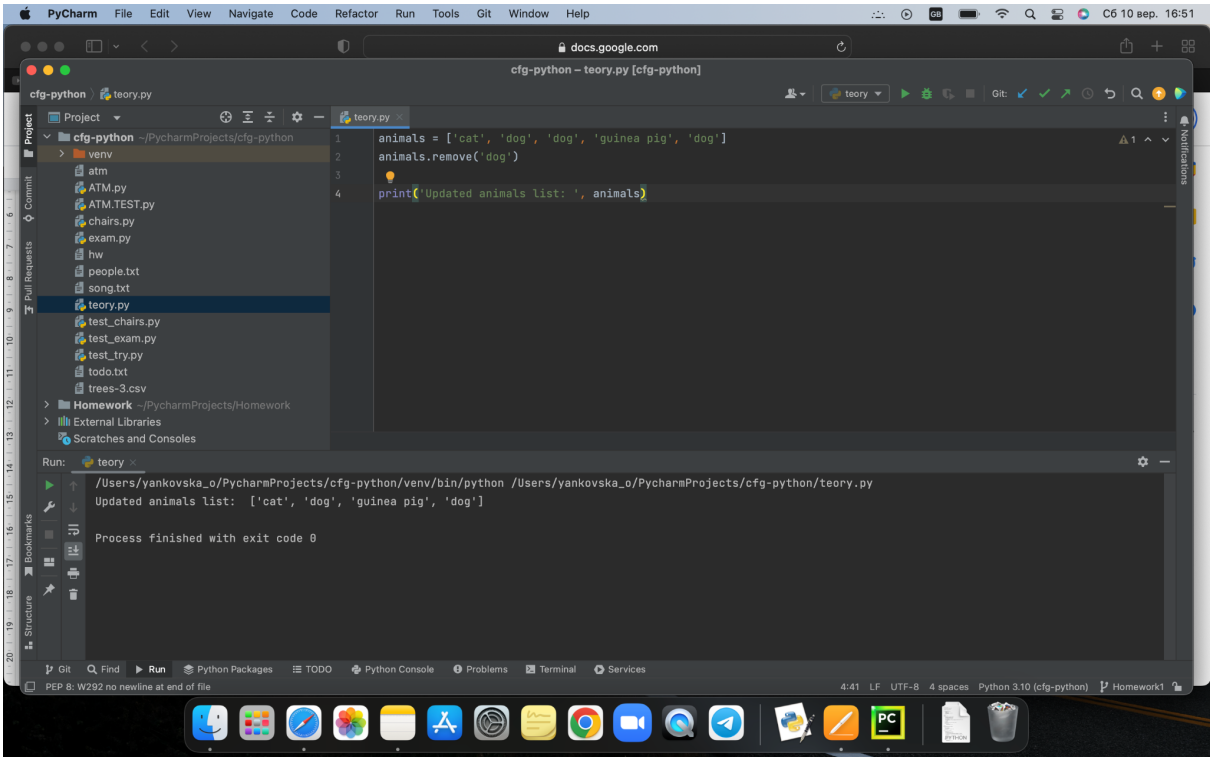


---

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

The remove() method takes a single element as an argument and removes it from the list.
If the element doesn't exist, it throws ValueError: list.remove(x): x not in list exception.

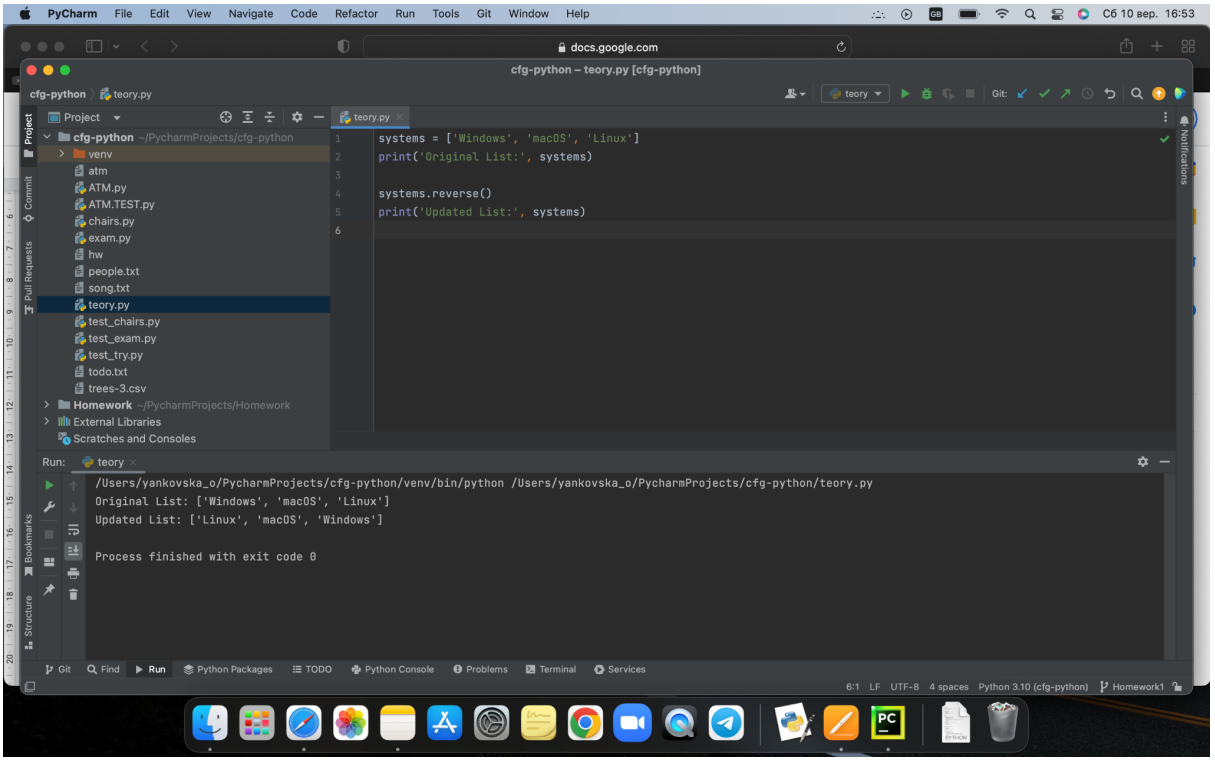
```
animals = ['cat', 'dog', 'dog', 'guinea pig', 'dog']
animals.remove('dog')

print('Updated animals list: ', animals)
```



---

reverse()
The reverse() method reverses the elements of the list.

```
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)

systems.reverse()
print('Updated List:', systems)
```

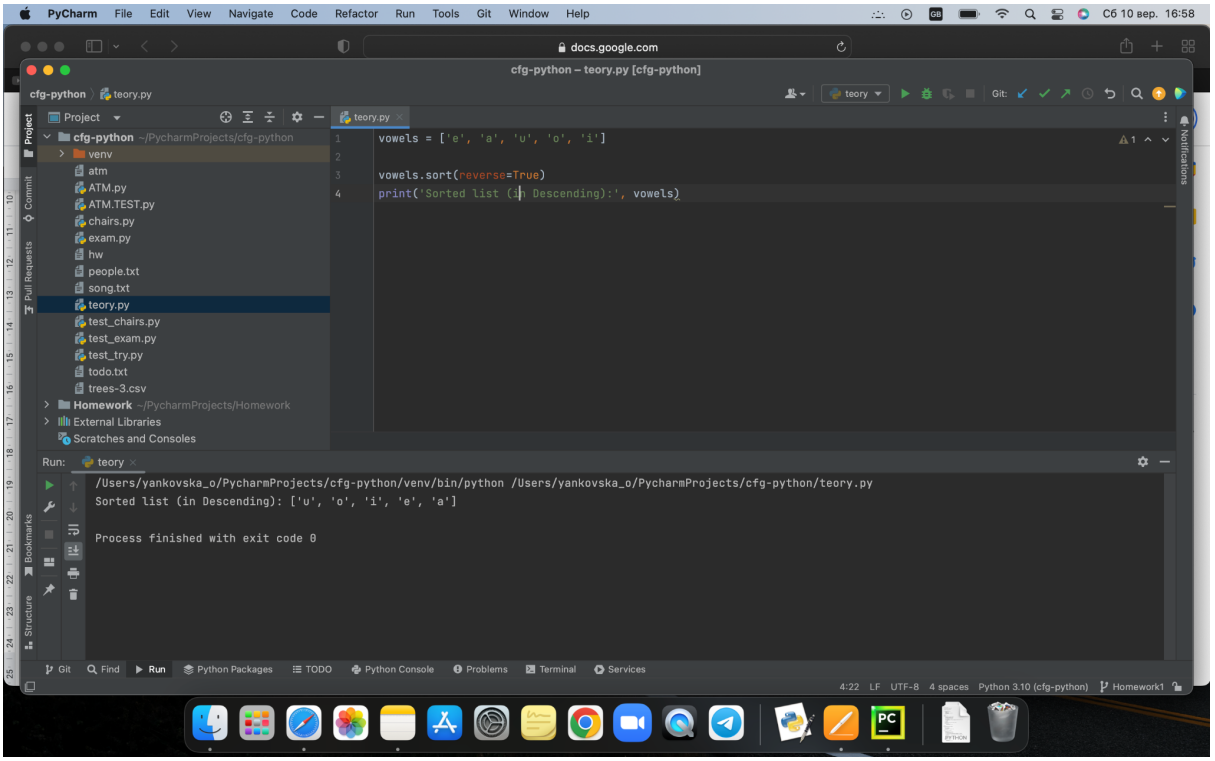


---

sort()
The sort() method sorts the items of a list in ascending or descending order.
sort() changes the list directly and doesn't return any value
The syntax of the sort() method is:
list.sort(key=..., reverse=...)

By default, sort() doesn't require any extra parameters. However, it 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
The sort() method accepts a reverse parameter as an optional argument.
Setting reverse = True sorts the list in the descending order.
list.sort(reverse=True)
EXAMPLE:
vowels = ['e', 'a', 'u', 'o', 'i']

vowels.sort(reverse=True)
print('Sorted list (in Descending):', vowels)

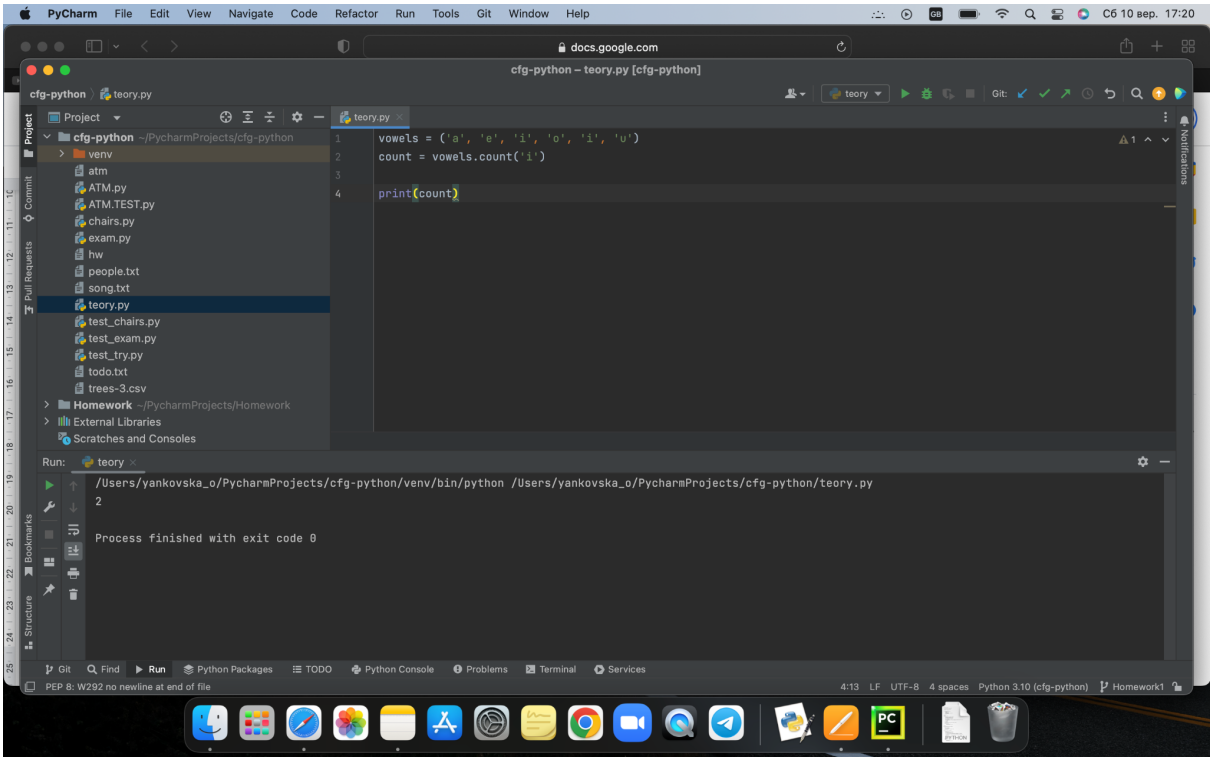


---

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

count()
The count() method returns the number of times the specified element appears in the tuple.

```
vowels = ('a', 'e', 'i', 'o', 'i', 'u')
count = vowels.count('i')

print(count)
```



---

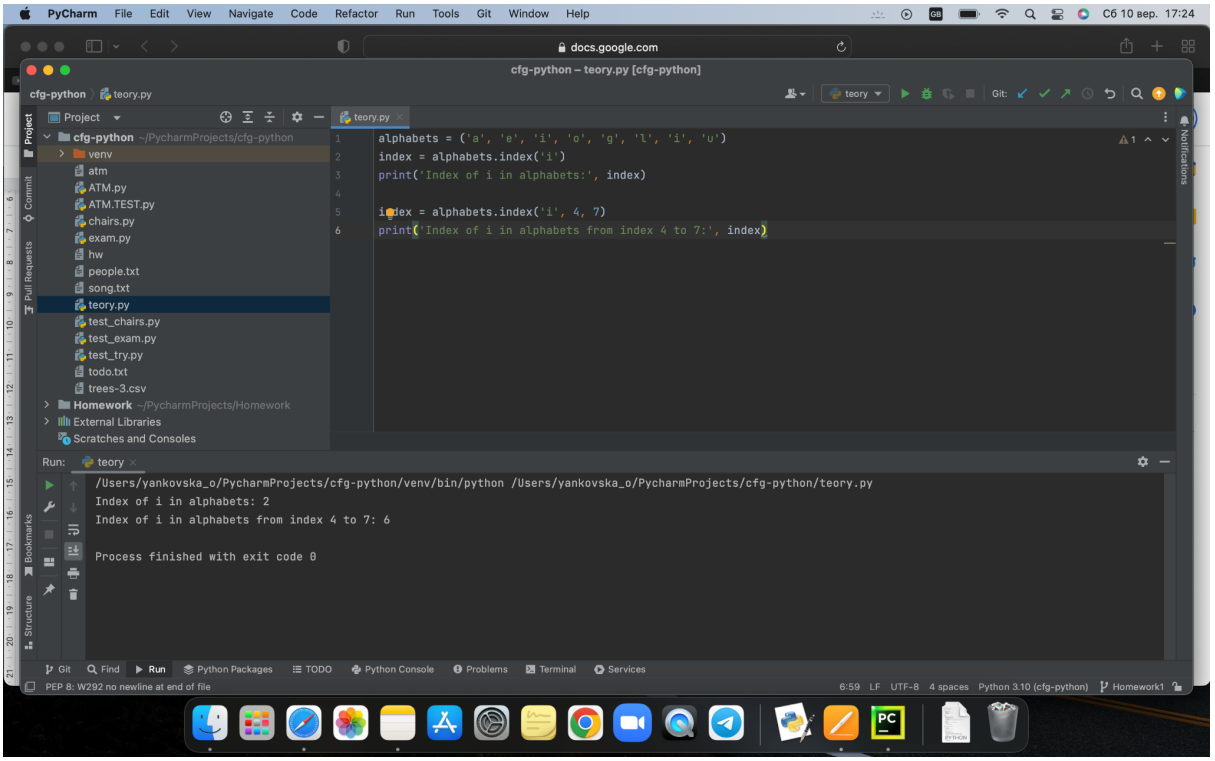
index()

The index() method returns the index of the specified element in the tuple.

The syntax of the index() method is:
tuple.index(element, start_index, end_index)
Here, the index() scans the element in the tuple from start_index to end_index.

EXAMPLE:

```
alphabets = ('a', 'e', 'i', 'o', 'g', 'l', 'i', 'u')
index = alphabets.index('i')
print('Index of i in alphabets:', index)

index = alphabets.index('i', 4, 7)
print('Index of i in alphabets from index 4 to 7:', index)
```



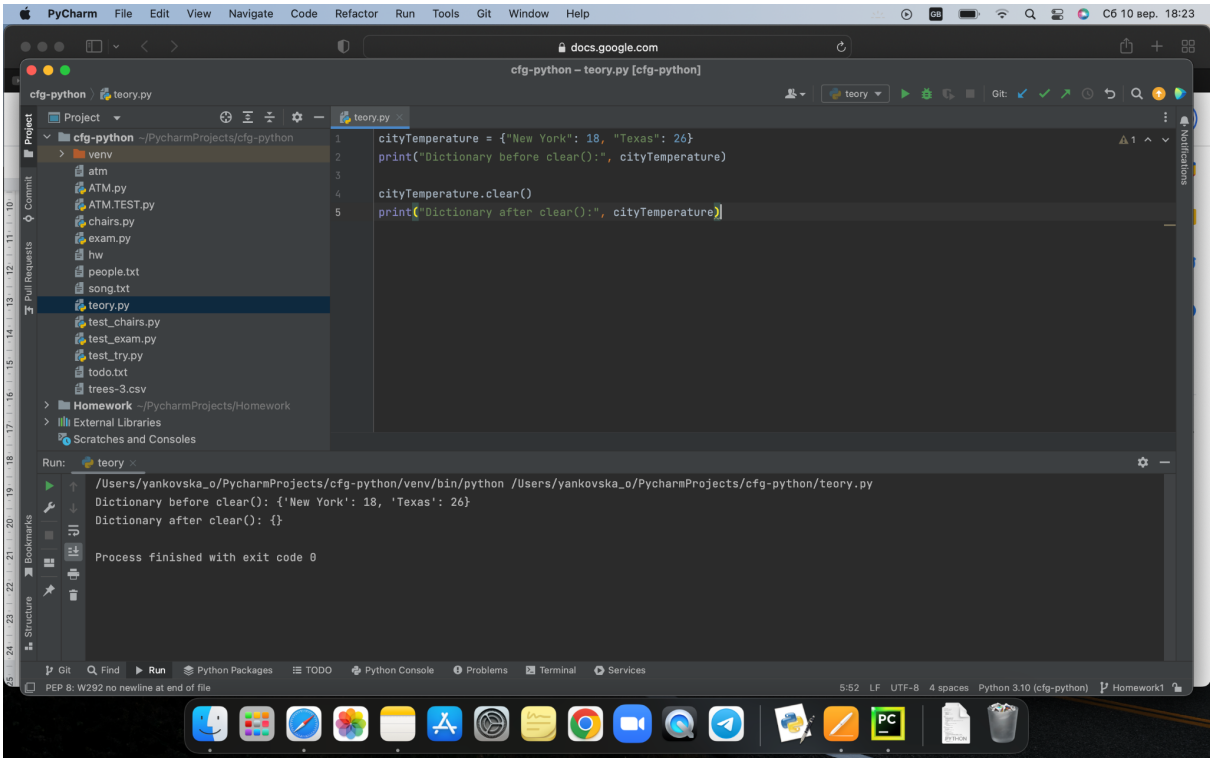
---

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

clear()

The clear() method removes all items from the dictionary.
The syntax of the clear() method is:
dictionary.clear()
EXAMPLE:

```
cityTemperature = {"New York": 18, "Texas": 26}
print("Dictionary before clear():", cityTemperature)

cityTemperature.clear()
print("Dictionary after clear():", cityTemperature)
```

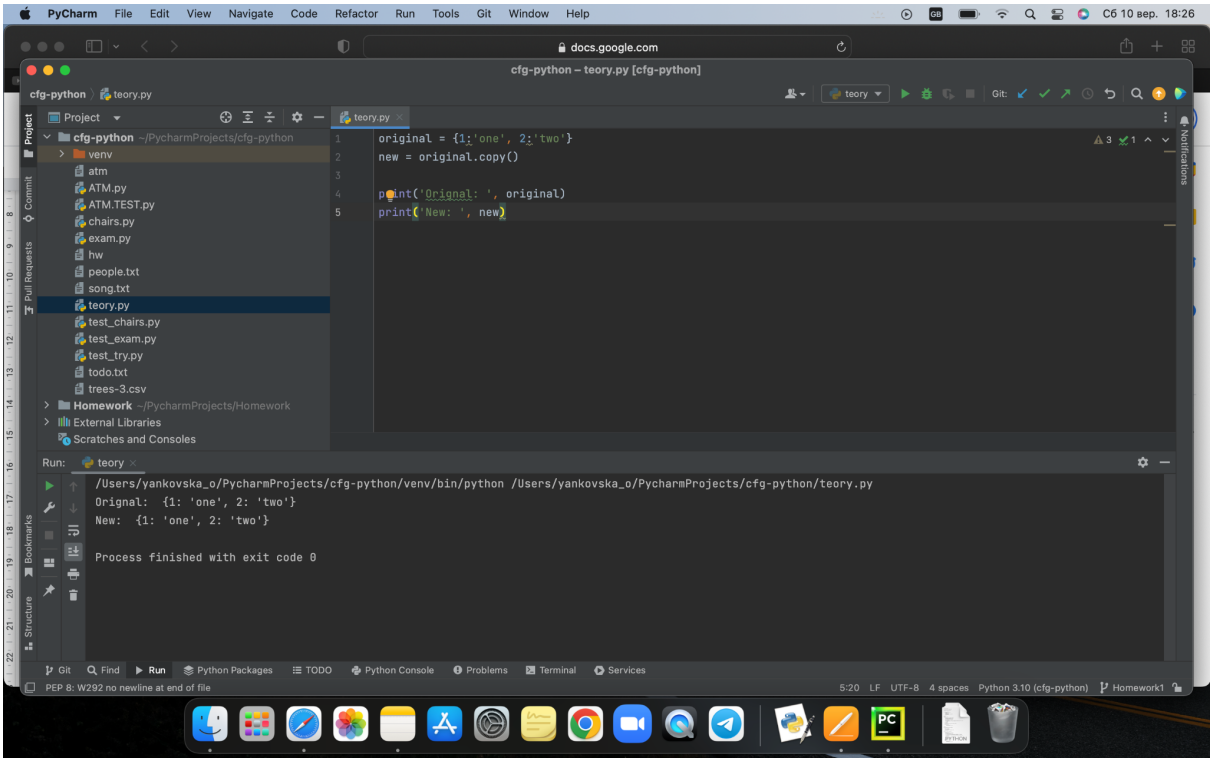


---

copy()

They copy() method returns a copy (shallow copy) of the dictionary.
The syntax of copy() is:
dict.copy()

```
original = {1:'one', 2:'two'}
new = original.copy()

print('Orignal: ', original)
print('New: ', new)
```



---

fromkeys()

The fromkeys() method creates a dictionary from the given sequence of keys and values.

The syntax of the fromkeys() method is:

dict.fromkeys(alphabets,number)

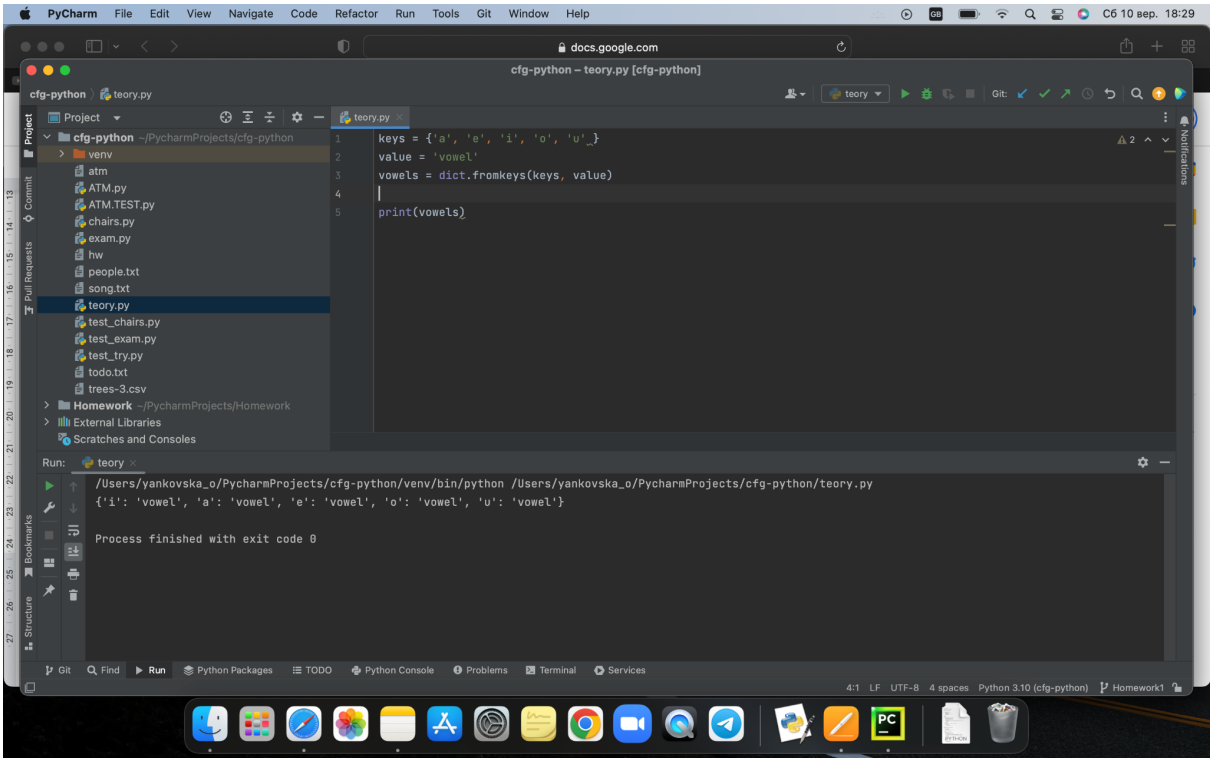
The fromkeys() method can take two parameters:

alphabets - are the keys that can be any iterables like string, set, list, etc.

numbers (Optional) - are the values that can be of any type or any iterables like string, set, list, etc.

EXAMPLE:

```
keys = {'a', 'e', 'i', 'o', 'u' }
value = 'vowel'
vowels = dict.fromkeys(keys, value)

print(vowels)
```



---

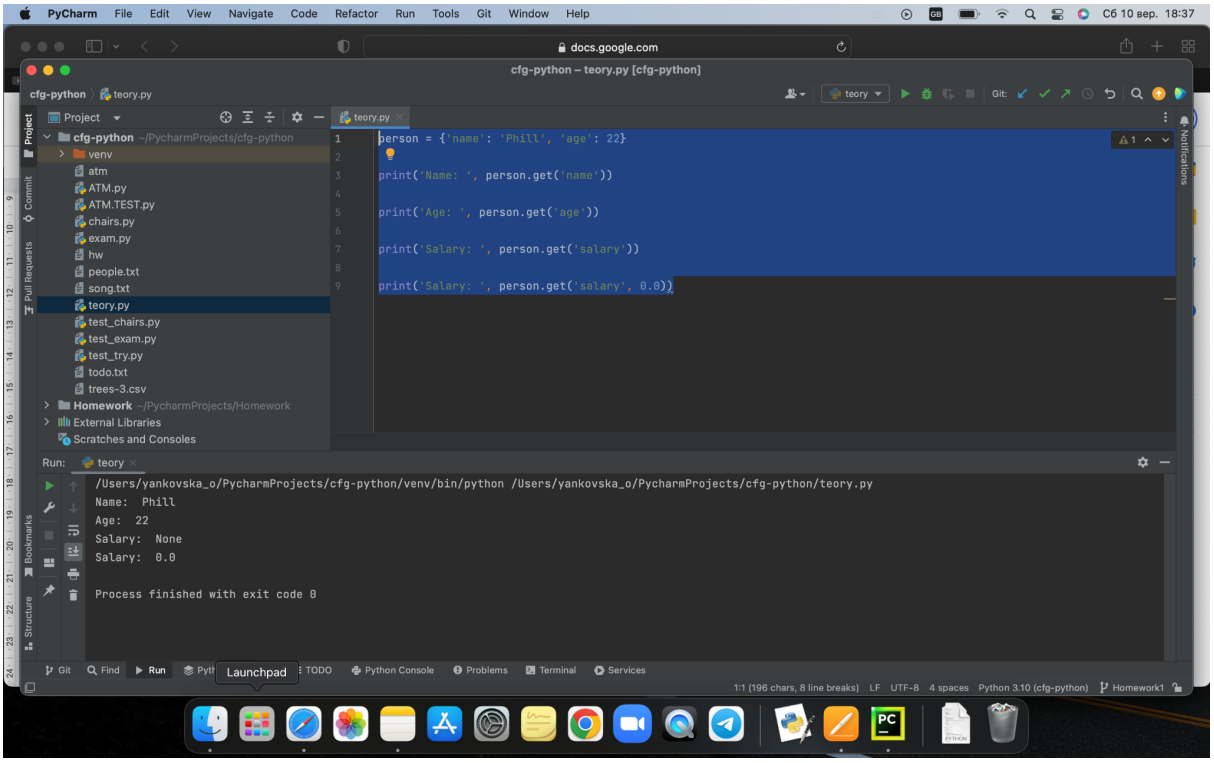
get()

The get() method returns the value for the specified key if the key is in the dictionary.

The syntax of get() is:
dict.get(key[, value])

get() method takes maximum of two parameters:
key - key to be searched in the dictionary
value (optional) - Value to be returned if the key is not found. The default value is None.

```
person = {'name': 'Phill', 'age': 22}

print('Name: ', person.get('name'))

print('Age: ', person.get('age'))

print('Salary: ', person.get('salary'))

print('Salary: ', person.get('salary', 0.0))
```



---

items()

The items() method returns a view object that displays a list of dictionary's (key, value) tuple pairs.

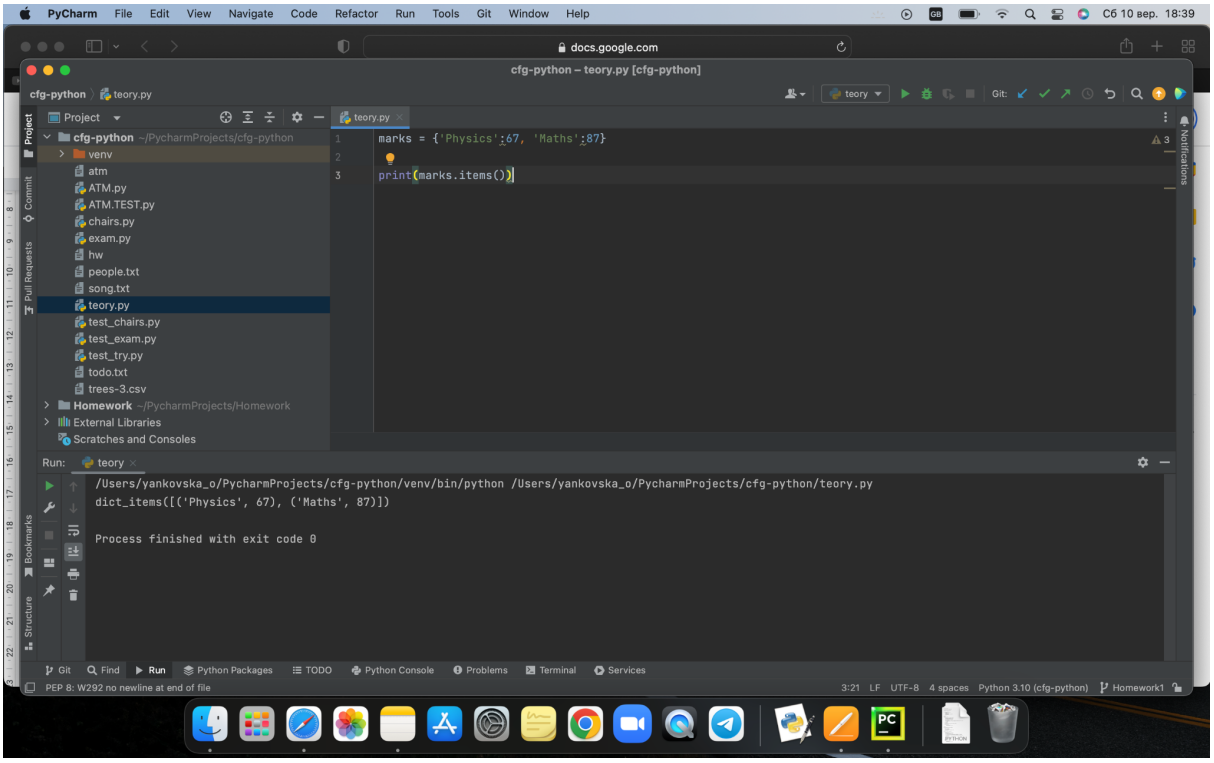
The syntax of items() method is:

dictionary.items()

EXAMPLE:

marks = {'Physics':67, 'Maths':87}

print(marks.items())



_____

keys()

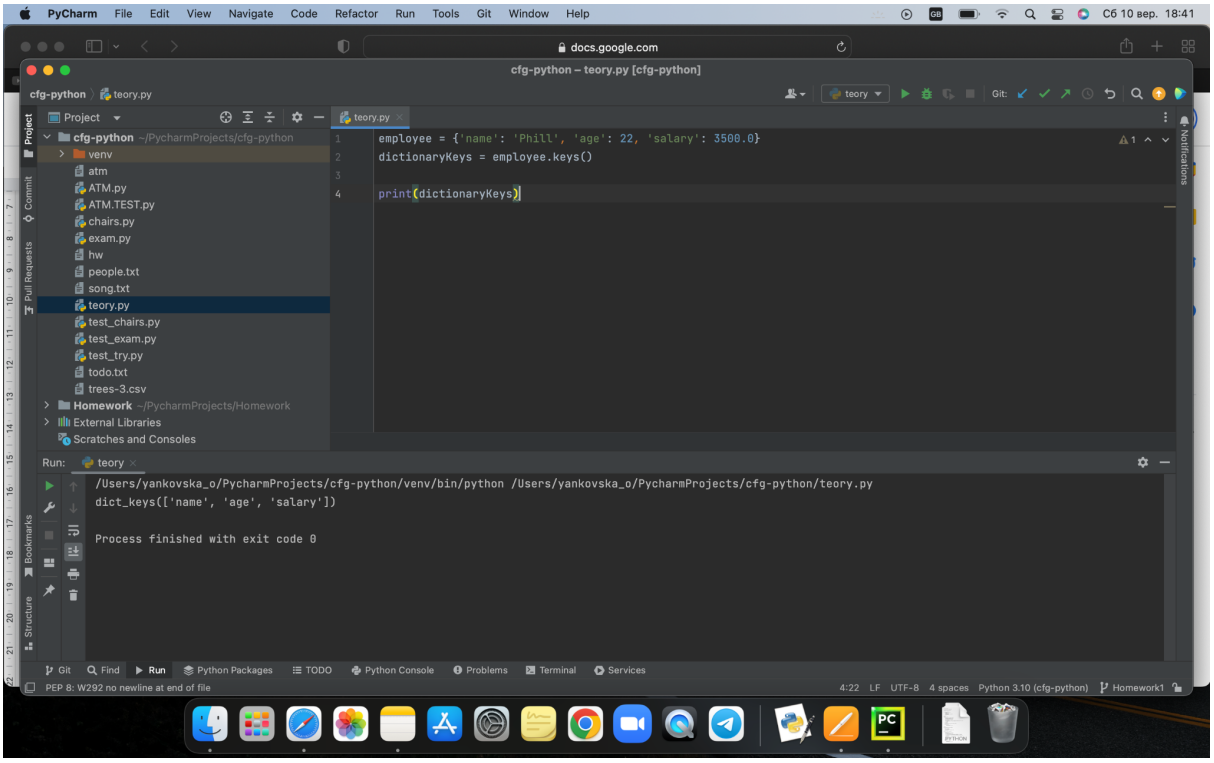
The keys() method extracts the keys of the dictionary and returns the list of keys as a view object.

The syntax of the keys() method is:

dict.keys()

EXAMPLE:

```
employee = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
dictionaryKeys = employee.keys()

print(dictionaryKeys)
```



---

pop()

The pop() method removes and returns an element from a dictionary having the given key.

The syntax of pop() method is

dictionary.pop(key[, default])

key - key which is to be searched for removal

default - value which is to be returned when the key is not in the dictionary

The pop() method returns:

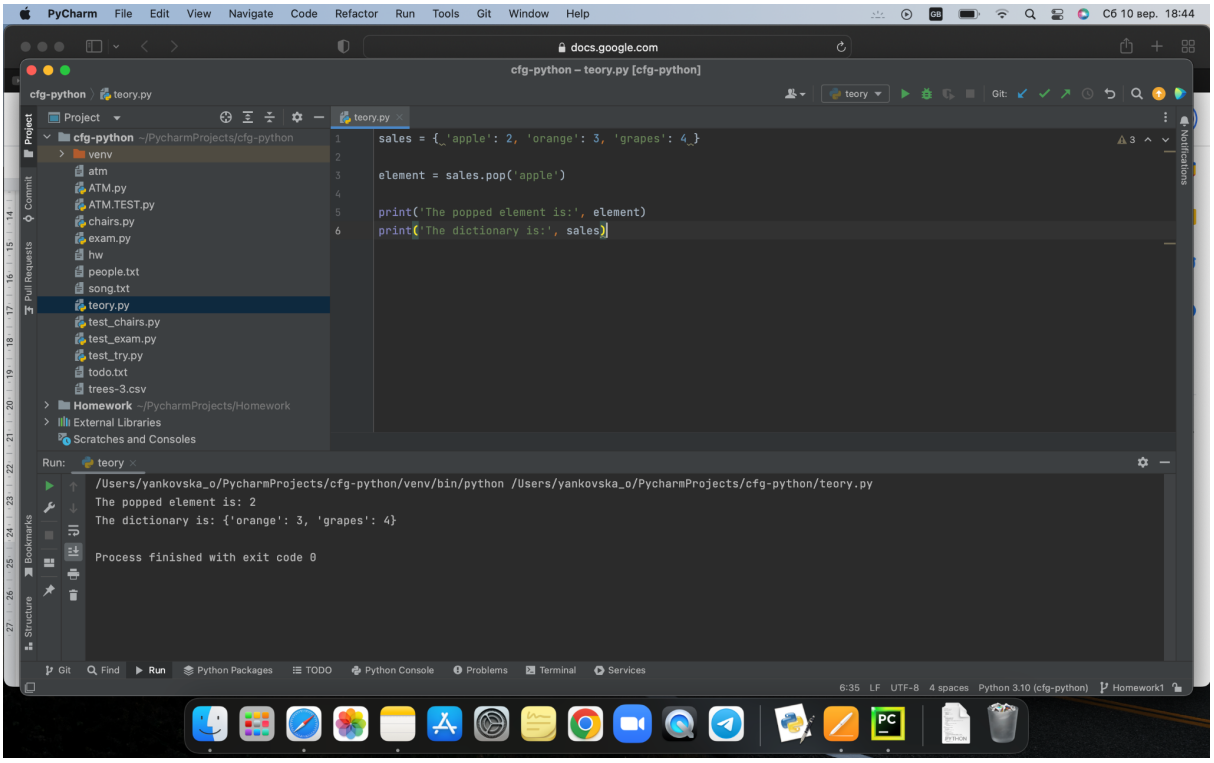
If key is found - removed/popped element from the dictionary

If key is not found - value specified as the second argument (default)

If key is not found and default argument is not specified - KeyError exception is raised

EXAMPLE:

```
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }

element = sales.pop('apple')

print('The popped element is:', element)
print('The dictionary is:', sales)
```



---

popitem()

The Python popitem() method removes and returns the last element (key, value) pair inserted into the dictionary.

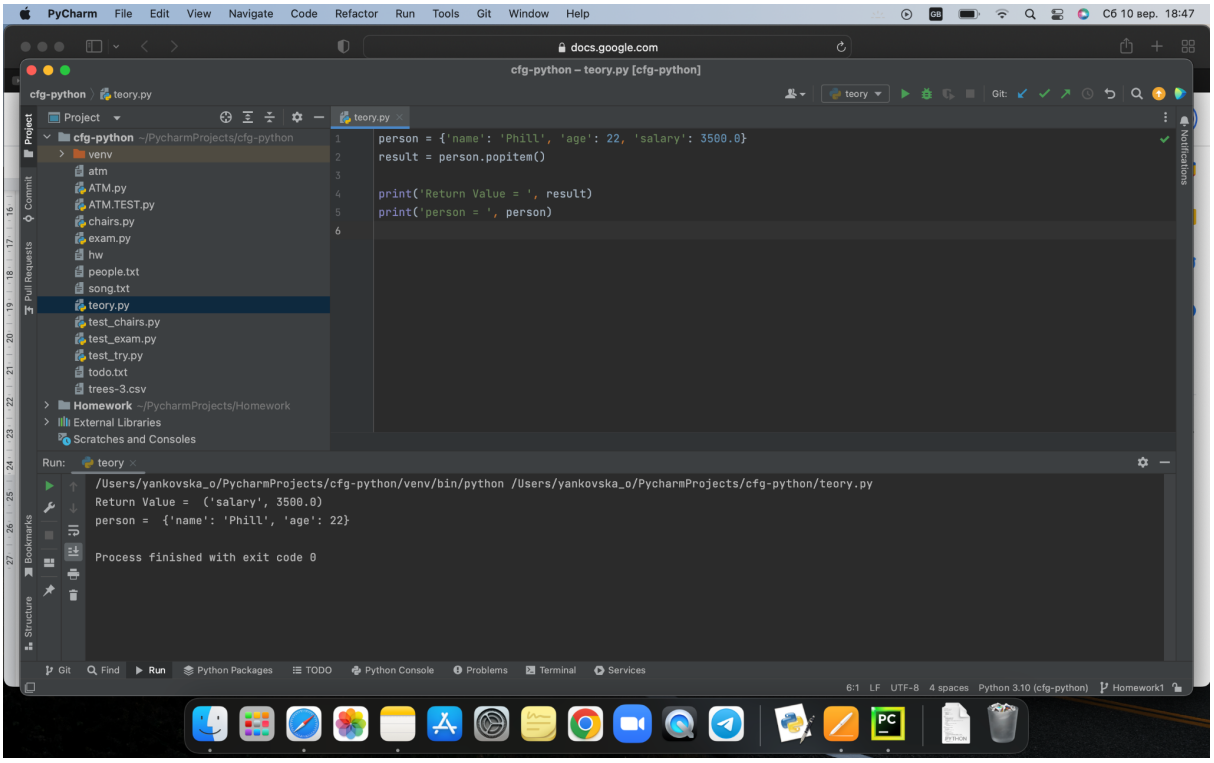
The syntax of popitem() is:

dict.popitem()

The popitem() method removes and returns the (key, value) pair from the dictionary in the Last In, First Out (LIFO) order.

Returns the latest inserted element (key,value) pair from the dictionary.
Removes the returned element pair from the dictionary.
EXAMPLE:

```
person = {'name': 'Phill', 'age': 22, 'salary': 3500.0}
result = person.popitem()

print('Return Value = ', result)
print('person = ', person)
```



---

setdefault()

The setdefault() method returns the value of a key (if the key is in dictionary). If not, it inserts key with a value to the dictionary.

The syntax of setdefault() is:

dict.setdefault(key[, default_value])

key - the key to be searched in the dictionary

default_value (optional) - key with a value default_value is inserted to the dictionary if the key is not in the dictionary.

If not provided, the default_value will be None.

setdefault() returns:

value of the key if it is in the dictionary

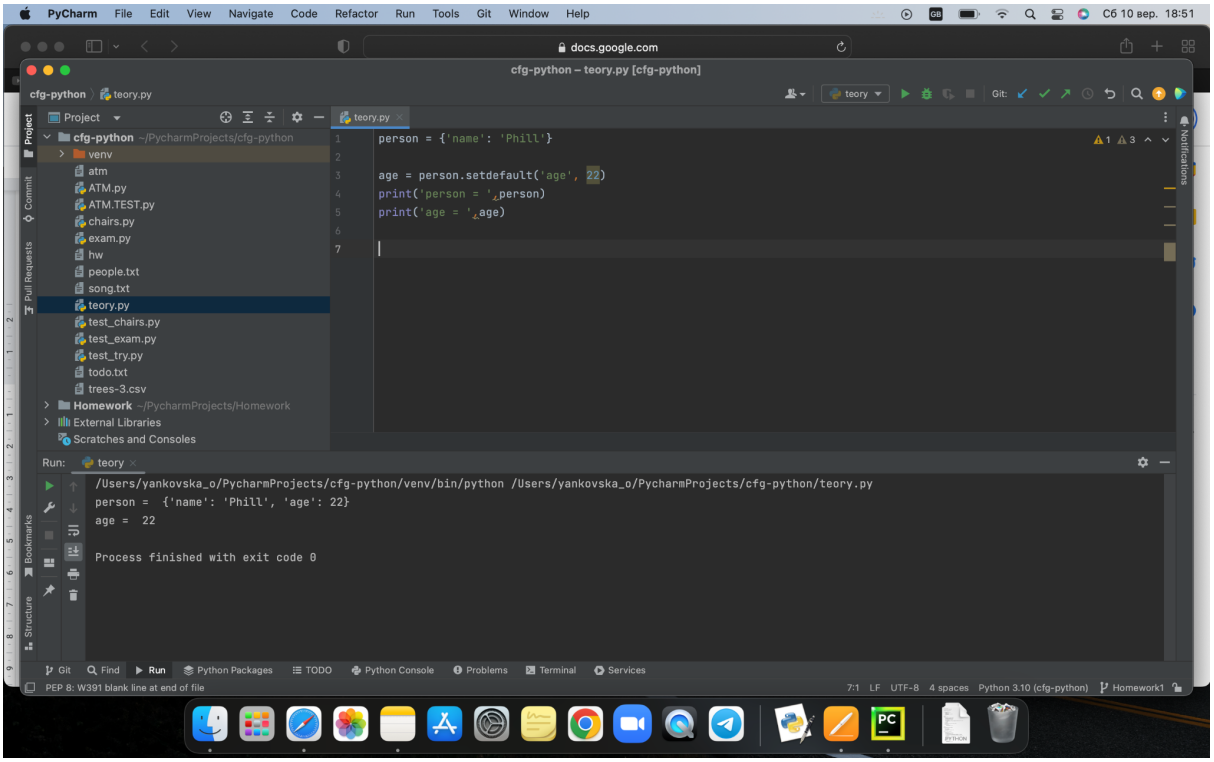
None if the key is not in the dictionary and default_value is not specified

default_value if key is not in the dictionary and default_value is specified

example:

```
person = {'name': 'Phill'}

age = person.setdefault('age', 22)
print('person = ',person)
```

print('age = ',age)



---

update()

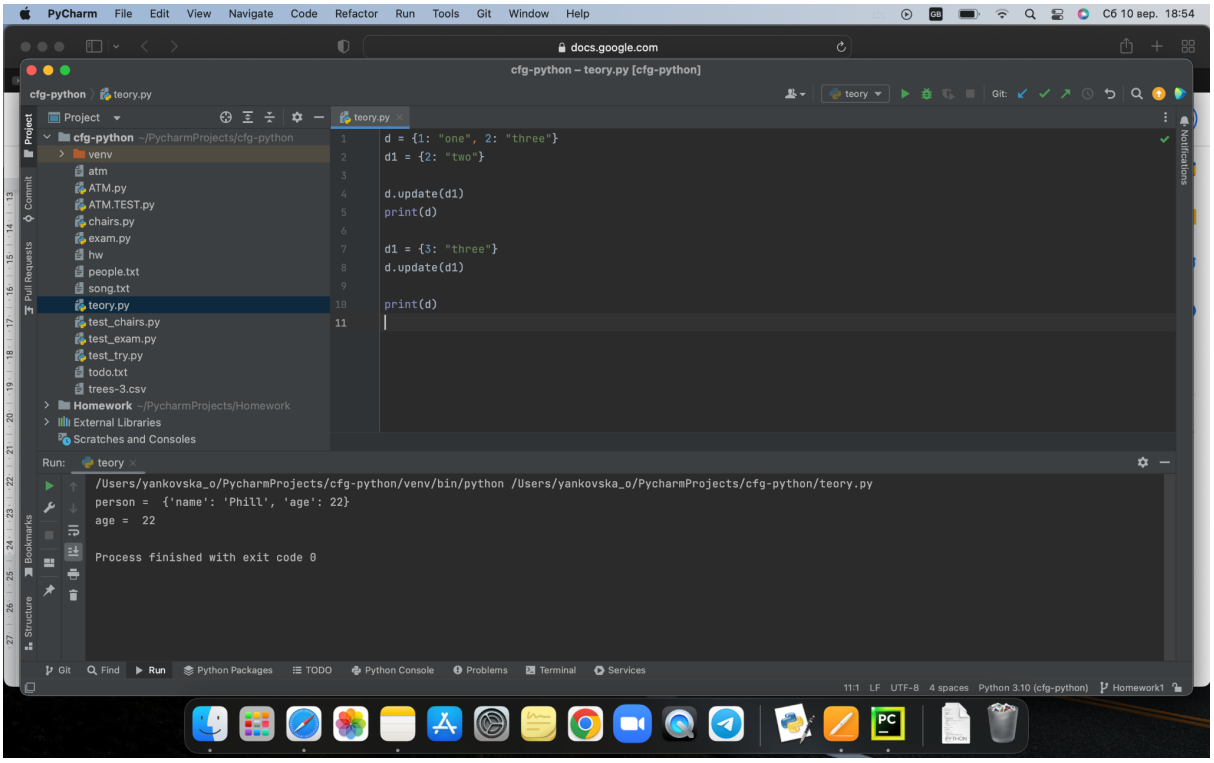
The update() method updates the dictionary with the elements from another dictionary object or from an iterable of key/value pairs.

The update() method takes either a dictionary or an iterable object of key/value pairs (generally tuples).

If update() is called without passing parameters, the dictionary remains unchanged.

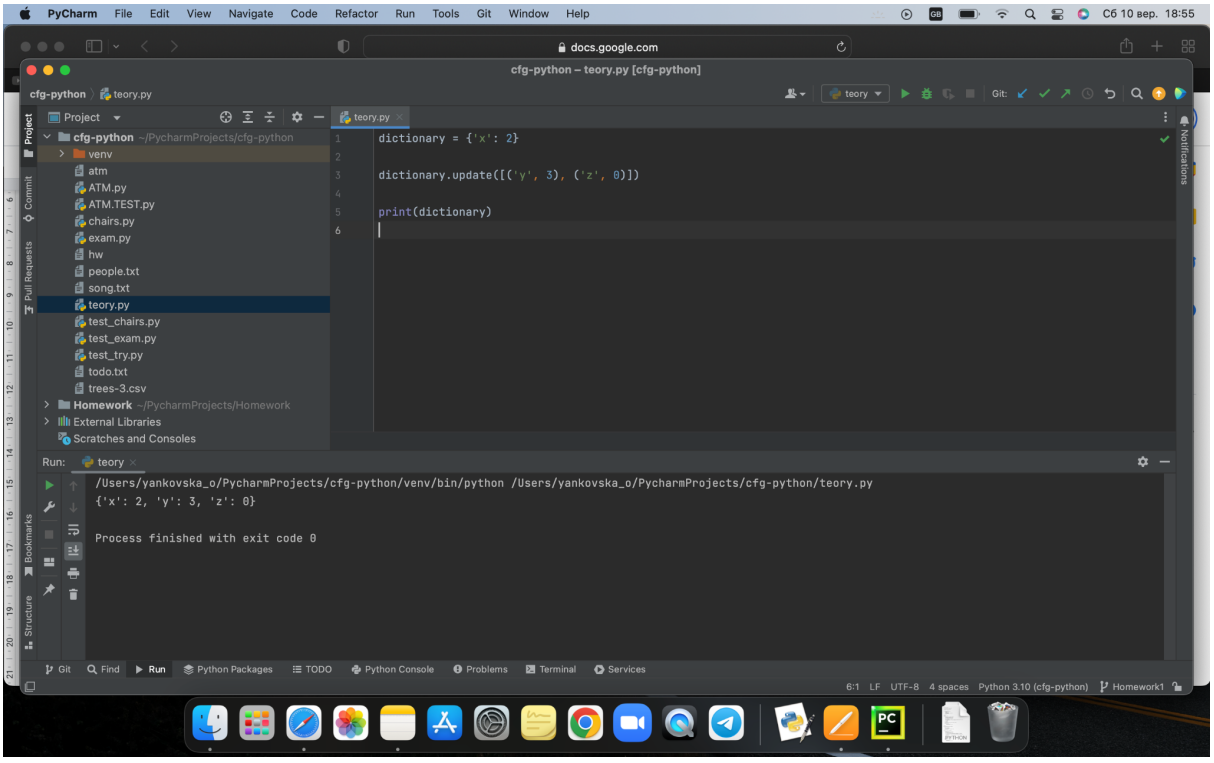
EXAMPLE:

```
d = {1: "one", 2: "three"}
d1 = {2: "two"}

d.update(d1)
print(d)

d1 = {3: "three"}
d.update(d1)

print(d)
```



dictionary = {'x': 2}
dictionary.update([('y', 3), ('z', 0)])

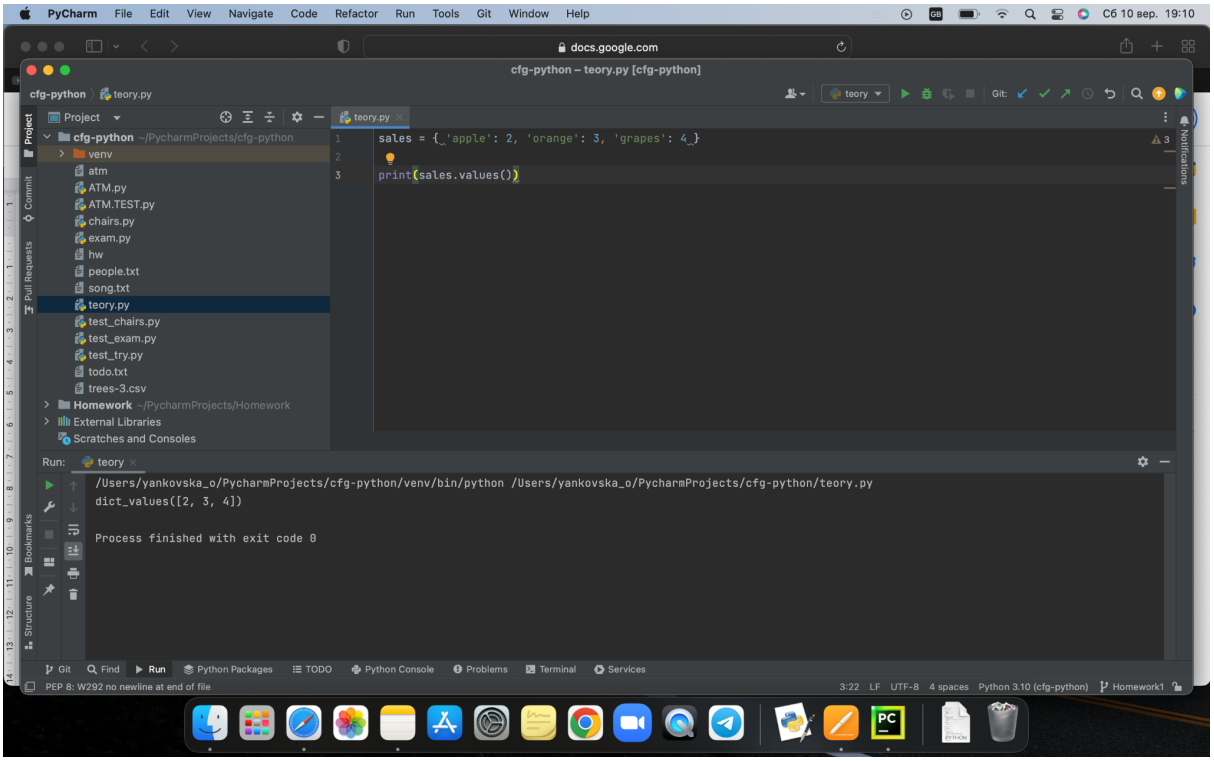
print(dictionary)



---

values()

The values() method returns a view object that displays a list of all the values in the dictionary.
The syntax of values() is:
dictionary.values()
EXAMPLE:
sales = { 'apple': 2, 'orange': 3, 'grapes': 4 }
print(sales.values())



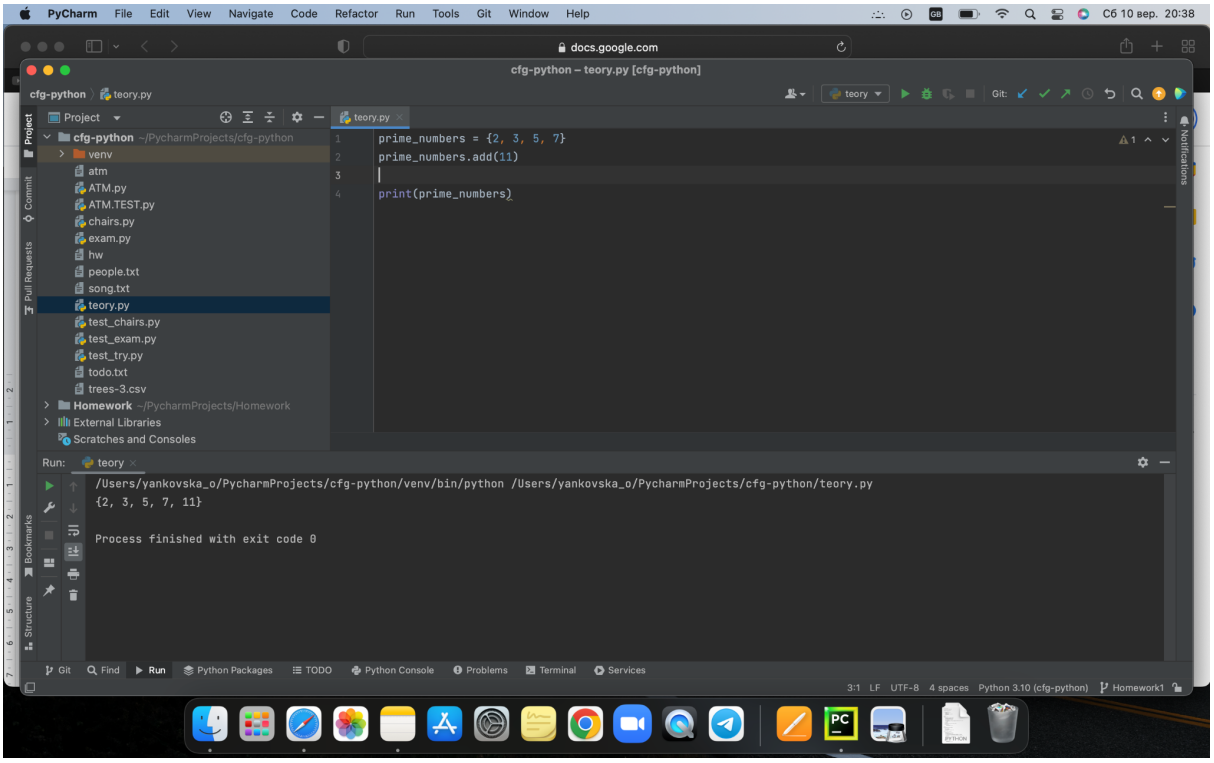
---

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

add()
The add() method adds a given element to a set. If the element is already present, it doesn't add any element.

The syntax of add() method is:
set.add(elem)
add() method doesn't add an element to the set if it's already present in it.
EXAMPLE:

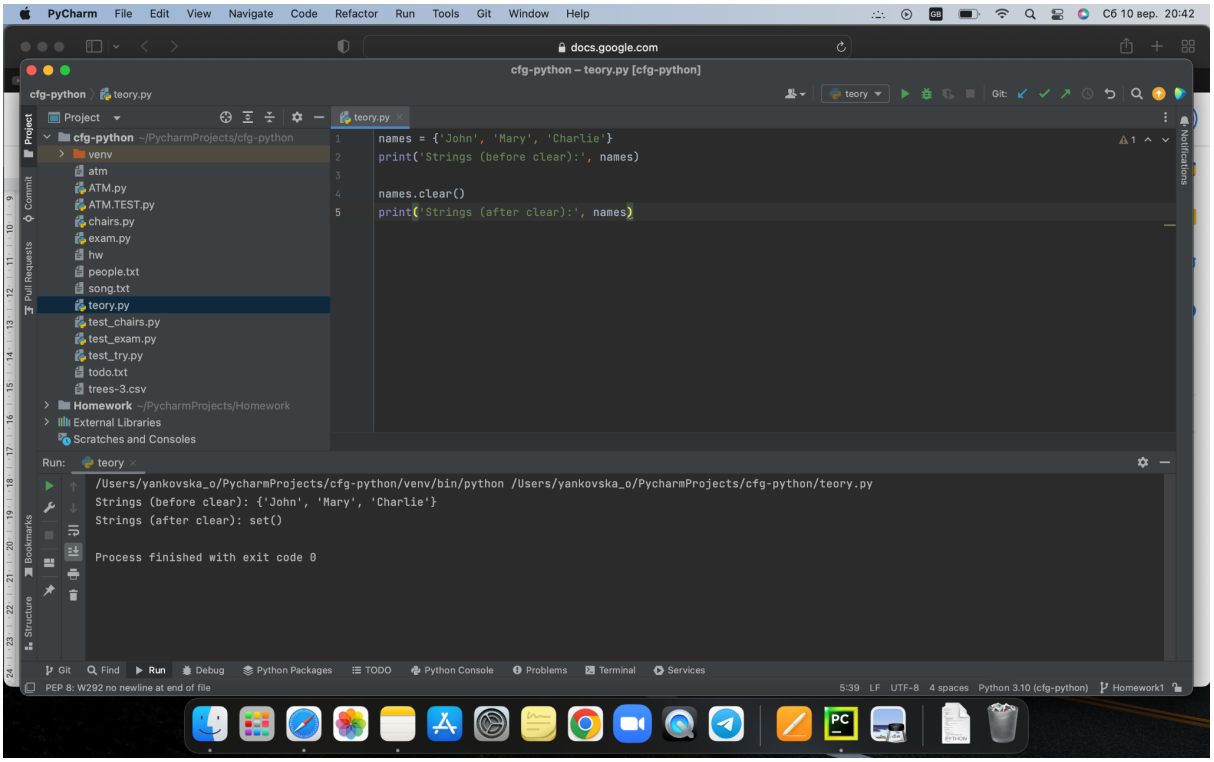
```
prime_numbers = {2, 3, 5, 7}
prime_numbers.add(11)

print(prime_numbers)
```



clear()
The clear() method removes all items from the set.

The syntax of the clear() method is:
set.clear()
EXAMPLE:
names = {'John', 'Mary', 'Charlie'}
print('Strings (before clear):', names)

names.clear()
print('Strings (after clear):', names)



copy()

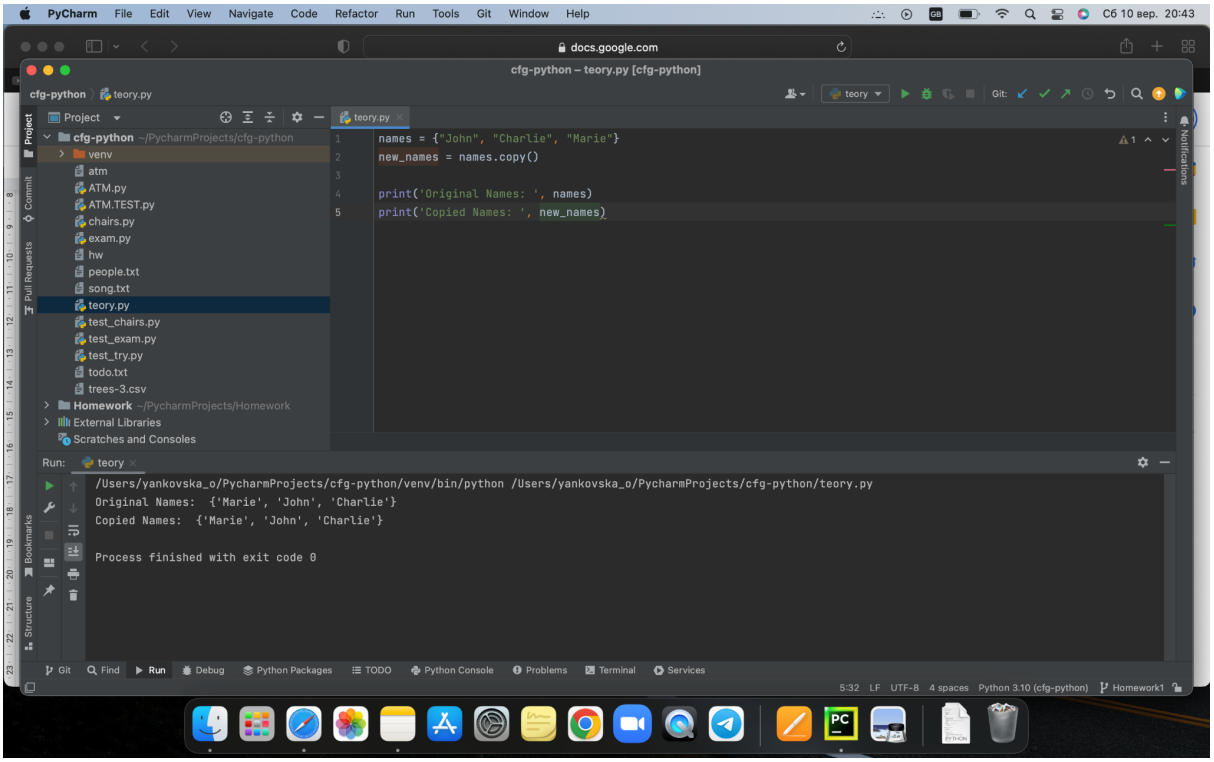
The copy() method returns a copy of the set.

The syntax of copy() method is:

set.copy()

EXAMPLE:

```
names = {"John", "Charlie", "Marie"}
new_names = names.copy()

print('Original Names: ', names)
print('Copied Names: ', new_names)
```



difference()

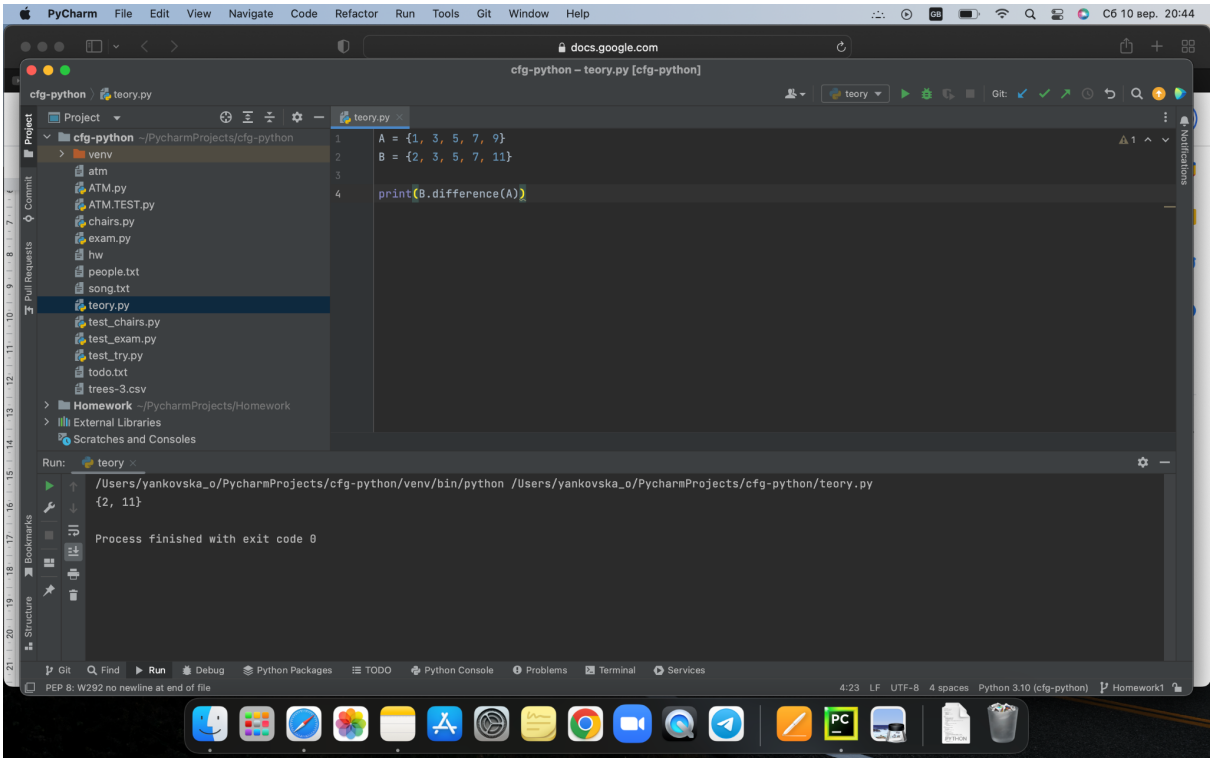
The difference() method computes the difference of two sets and returns items that are unique to the first set.

example:

A = {1, 3, 5, 7, 9}

B = {2, 3, 5, 7, 11}

print(A.difference(B))



intersection()
The intersection() method returns a new set with elements that are common to all sets.
The syntax of intersection() in Python is:
A.intersection(*other_sets)
If the argument is not passed to intersection(), it returns a shallow copy of the set (A).

EXAMPLE:
A = {2, 3, 5}
B = {1, 3, 5}

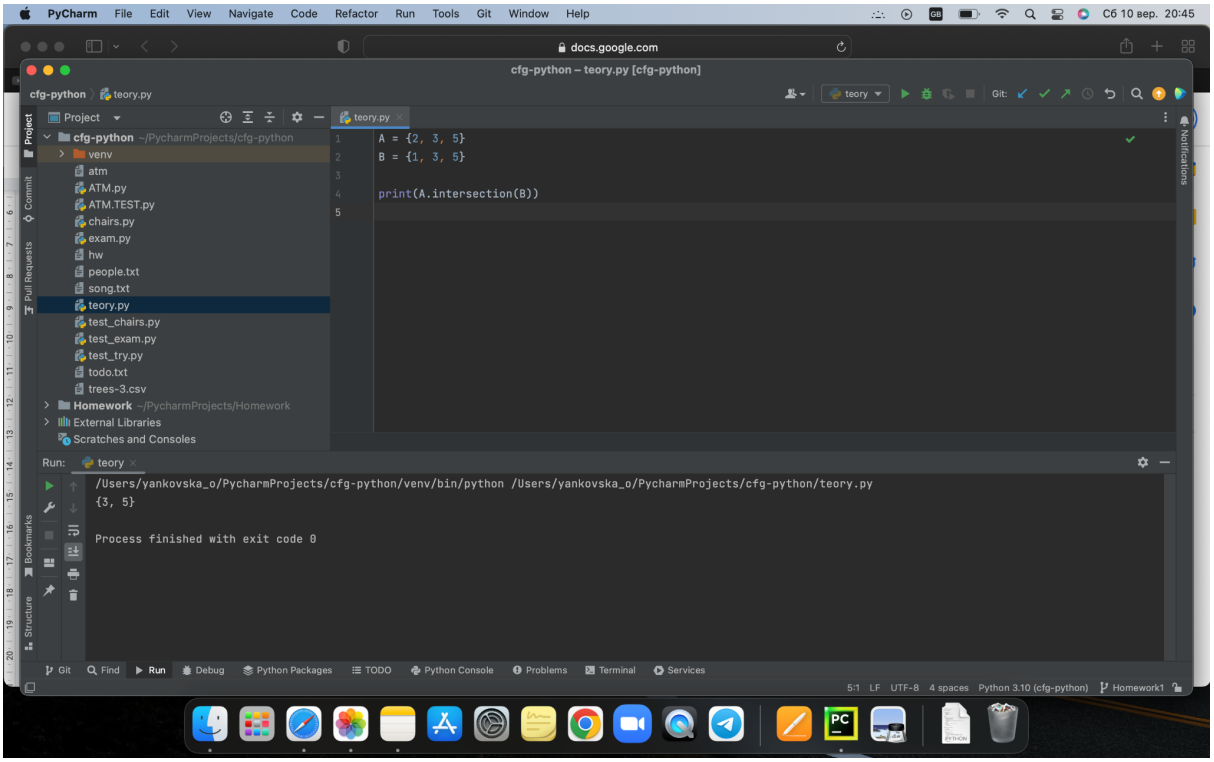
print(A.intersection(B))

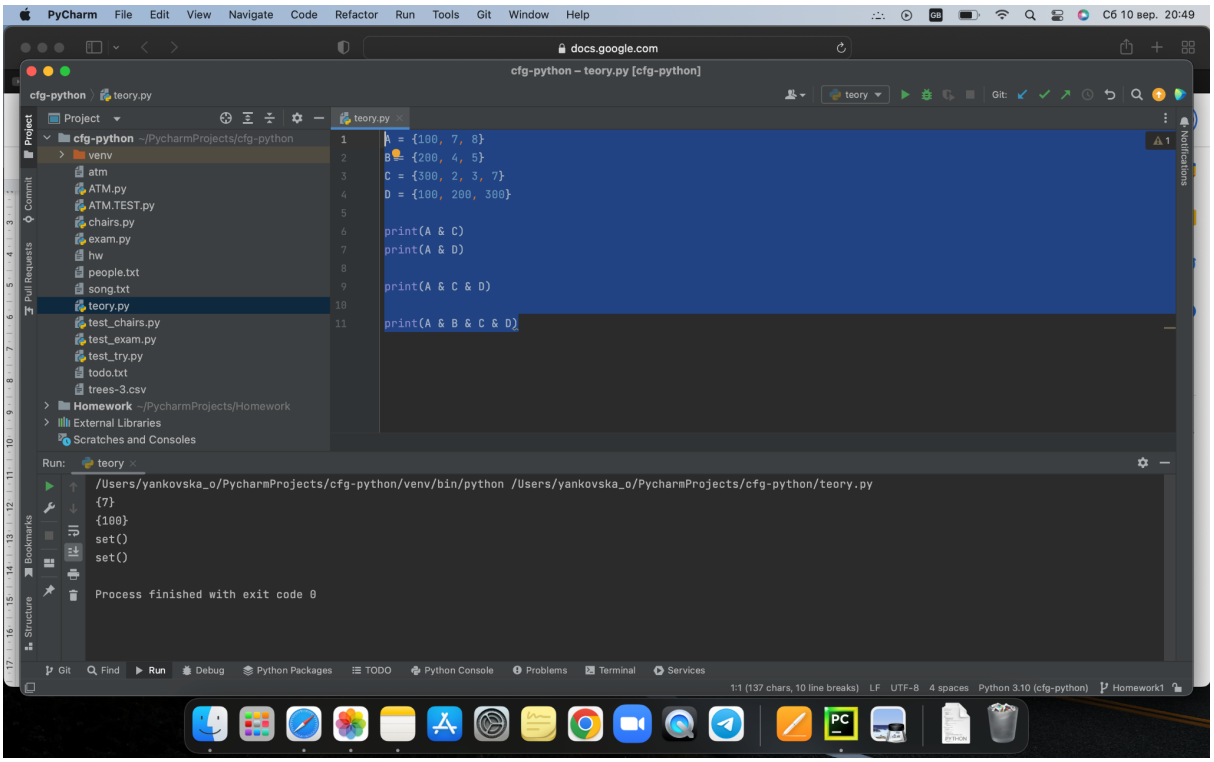
example2:
You can also find the intersection of sets using & operator.
A = {100, 7, 8}
B = {200, 4, 5}
C = {300, 2, 3, 7}
D = {100, 200, 300}

print(A & C)
print(A & D)

print(A & C & D)

print(A & B & C & D)

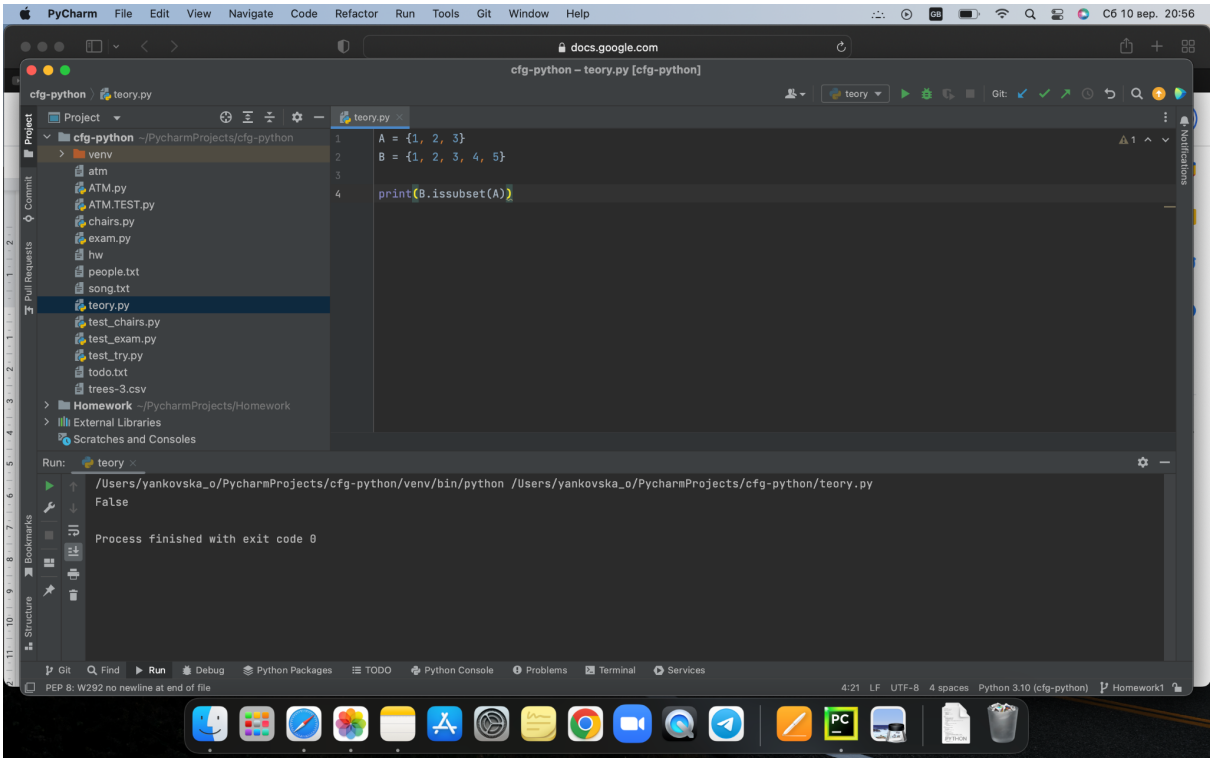




issubset()
The issubset() method returns True if set A is the subset of B, i.e. if all the elements of set A are present in set B . Else, it returns False.

EXAMPLE:
A = {1, 2, 3}
B = {1, 2, 3, 4, 5}

print(A.issubset(B))

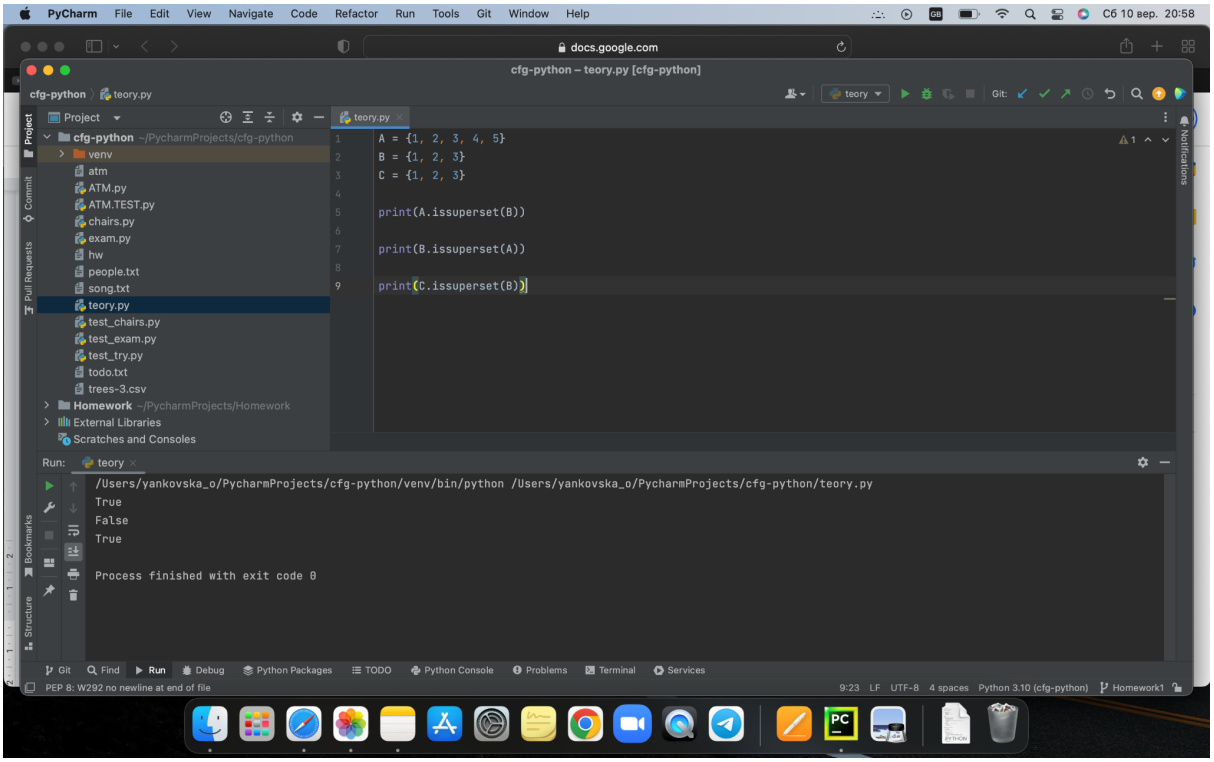


issuperset()
The issuperset() method returns True if a set has every elements of another set (passed as an argument). If not, it returns False.
example:
A = {1, 2, 3, 4, 5}
B = {1, 2, 3}
C = {1, 2, 3}

print(A.issuperset(B))

print(B.issuperset(A))

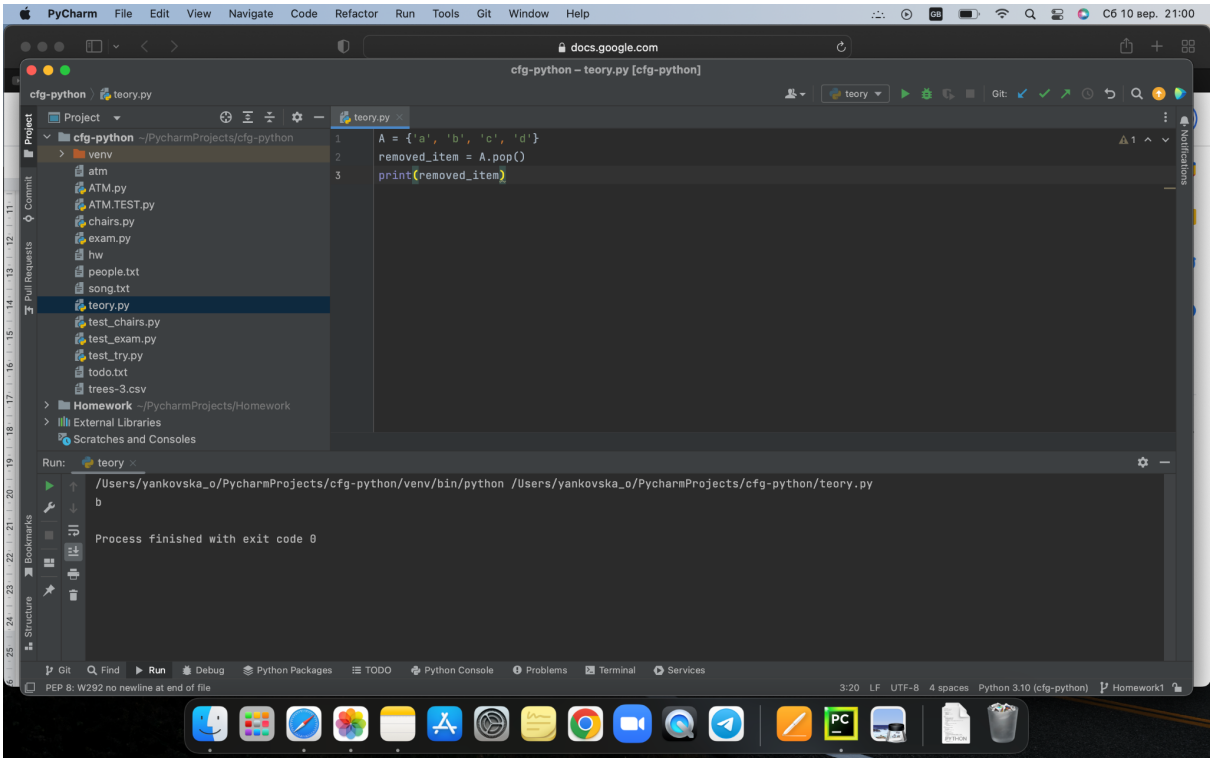
print(C.issuperset(B))



pop()

The pop() method randomly removes an item from a set and returns the removed item.
example:

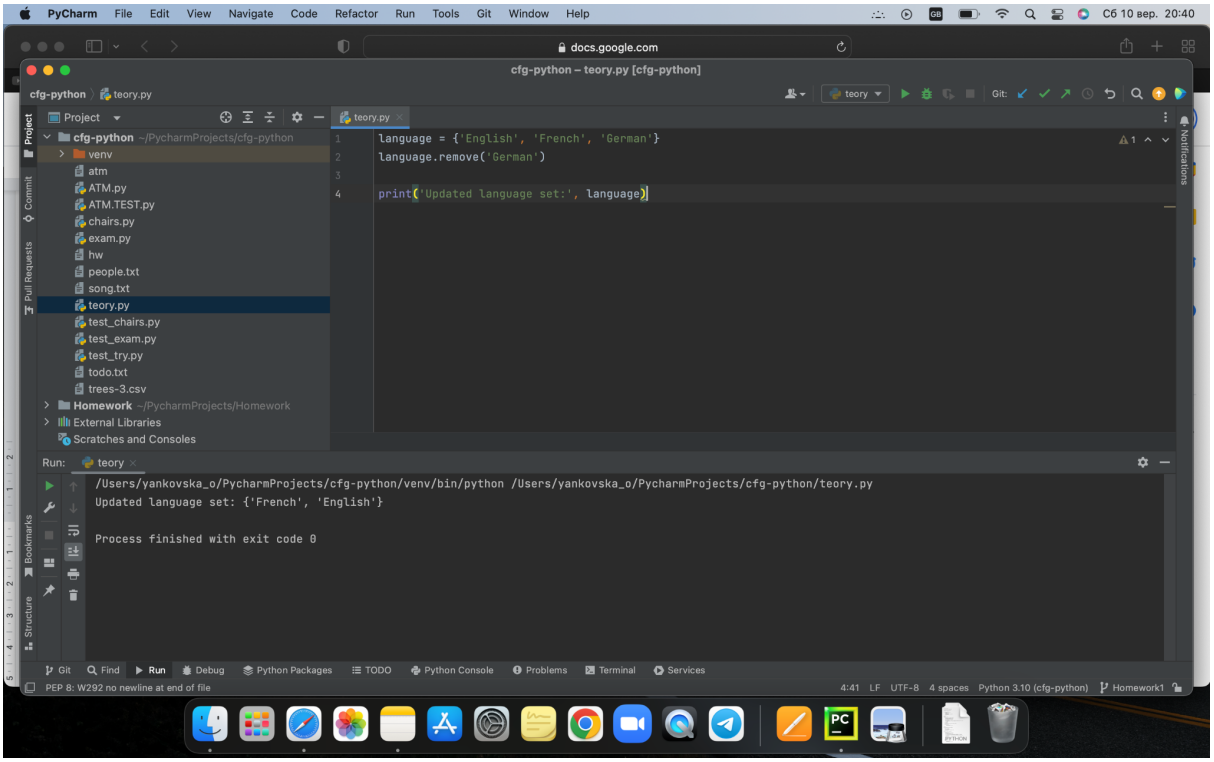
```
A = {'a', 'b', 'c', 'd'}
removed_item = A.pop()
print(removed_item)
```



remove()
The remove() method removes the specified element from the set.

The syntax of the remove() method is:
set.remove(element)
The remove() removes the specified element from the set and updates the set. It doesn't return any value.
If the element passed to remove() doesn't exist, KeyError exception is thrown.
EXAMPLE:
language = {'English', 'French', 'German'}
language.remove('German')

print('Updated language set:', language)



symmetric_difference()

The symmetric_difference() method returns all the items present in given sets, except the items in their intersections.

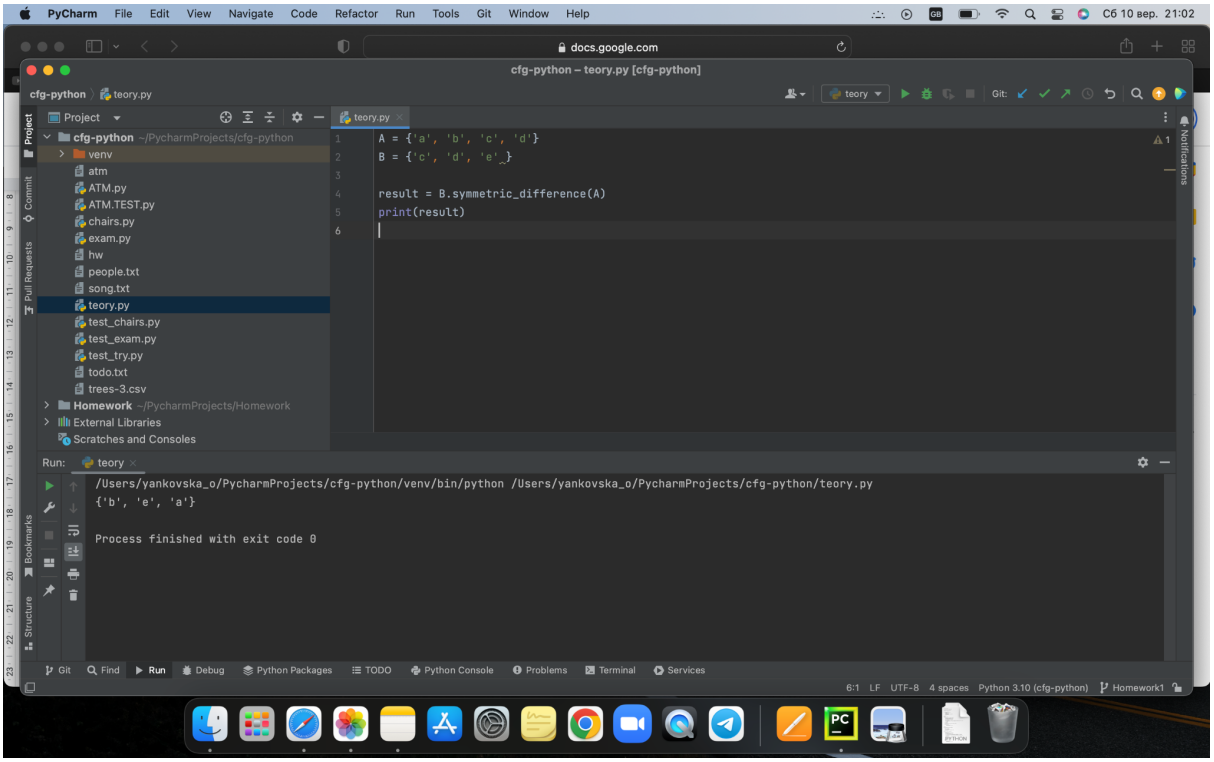
example:

A = {'a', 'b', 'c', 'd'}

B = {'c', 'd', 'e' }

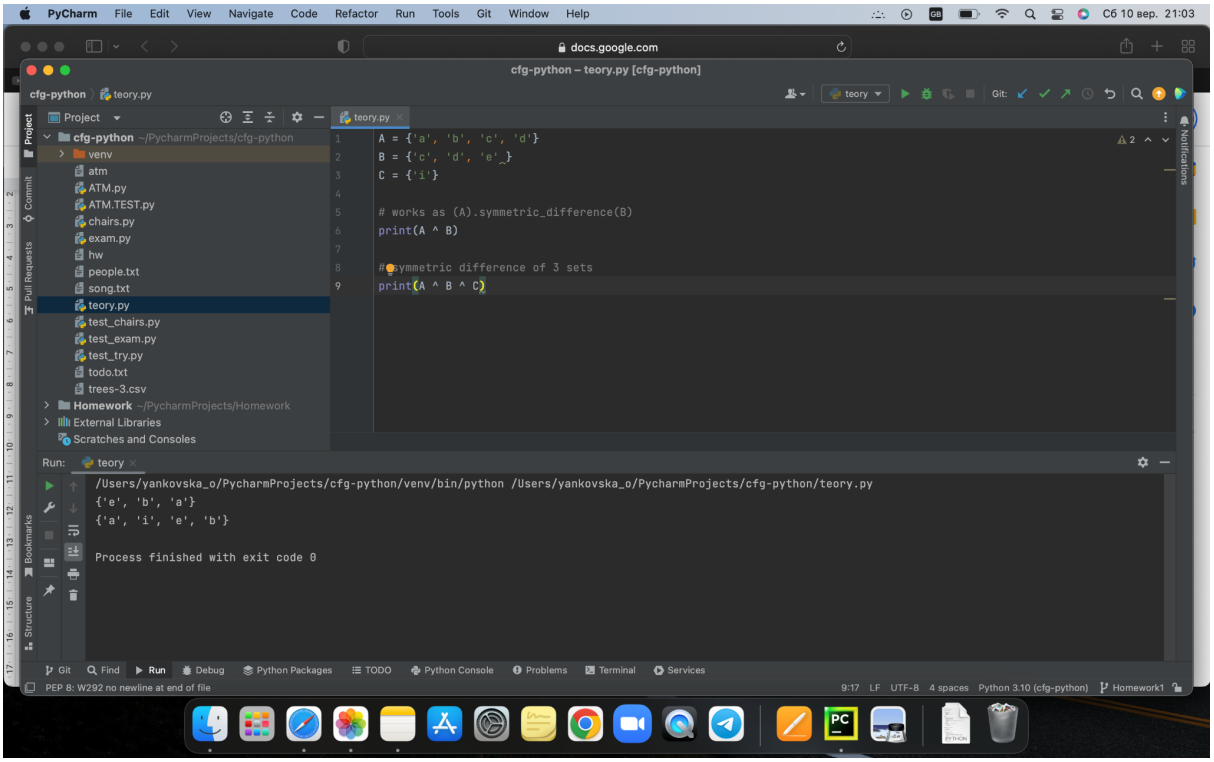
result = B.symmetric_difference(A)

print(result)



We can also find the symmetric difference using the ^ operator in Python. For example,

```
A = {'a', 'b', 'c', 'd'}
B = {'c', 'd', 'e' }
C = {'i'}

print(A ^ B)
print(A ^ B ^ C)
```



union()

The Python set union() method returns a new set with distinct elements from all the sets.

The syntax of union() is:

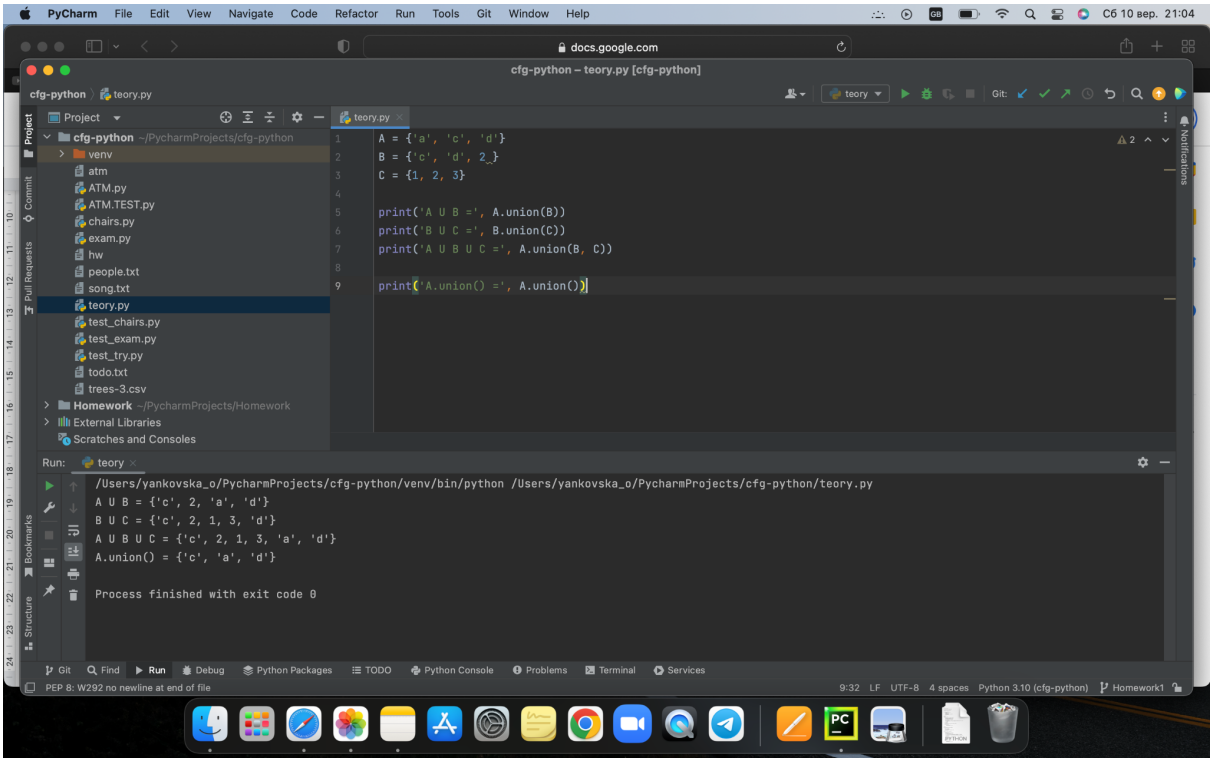
A.union(*other_sets)

EXAMPLE:

A = {'a', 'c', 'd'}
B = {'c', 'd', 2 }
C = {1, 2, 3}

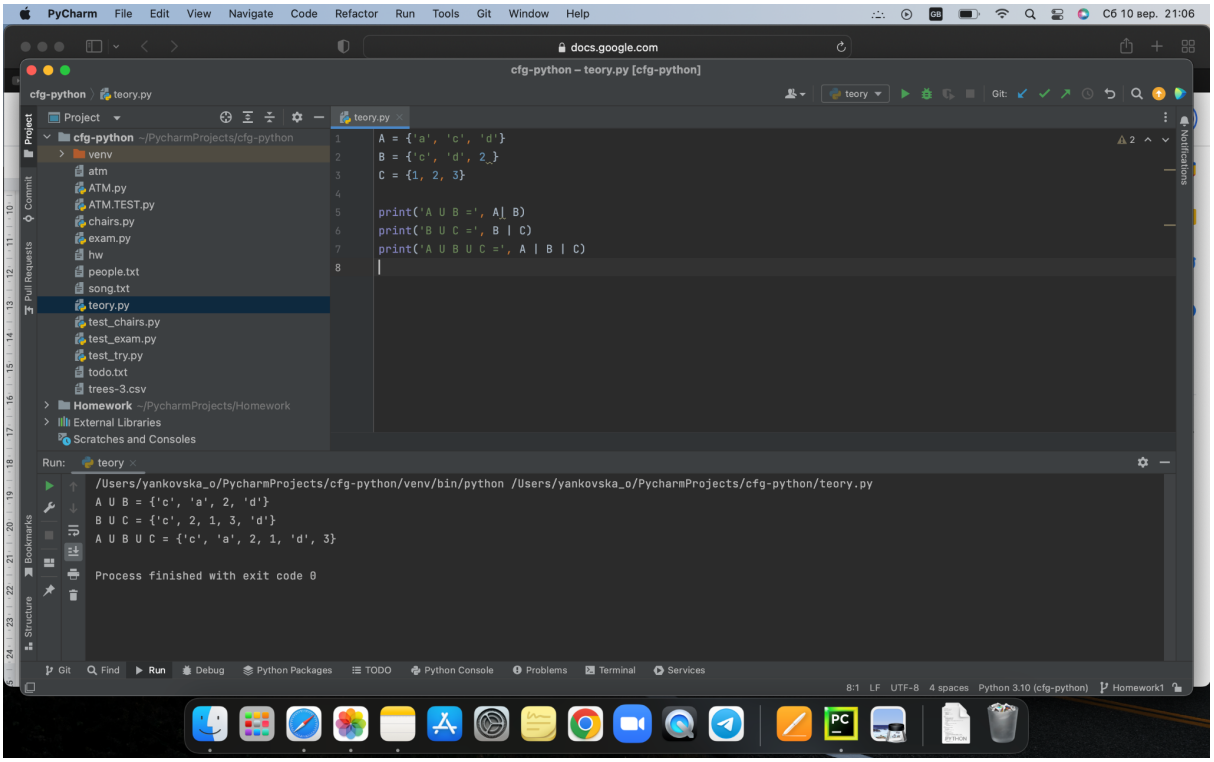
print('A U B =', A.union(B))
print('B U C =', B.union(C))
print('A U B U C =', A.union(B, C))

print('A.union() =', A.union())



You can also find the union of sets using the | operator.

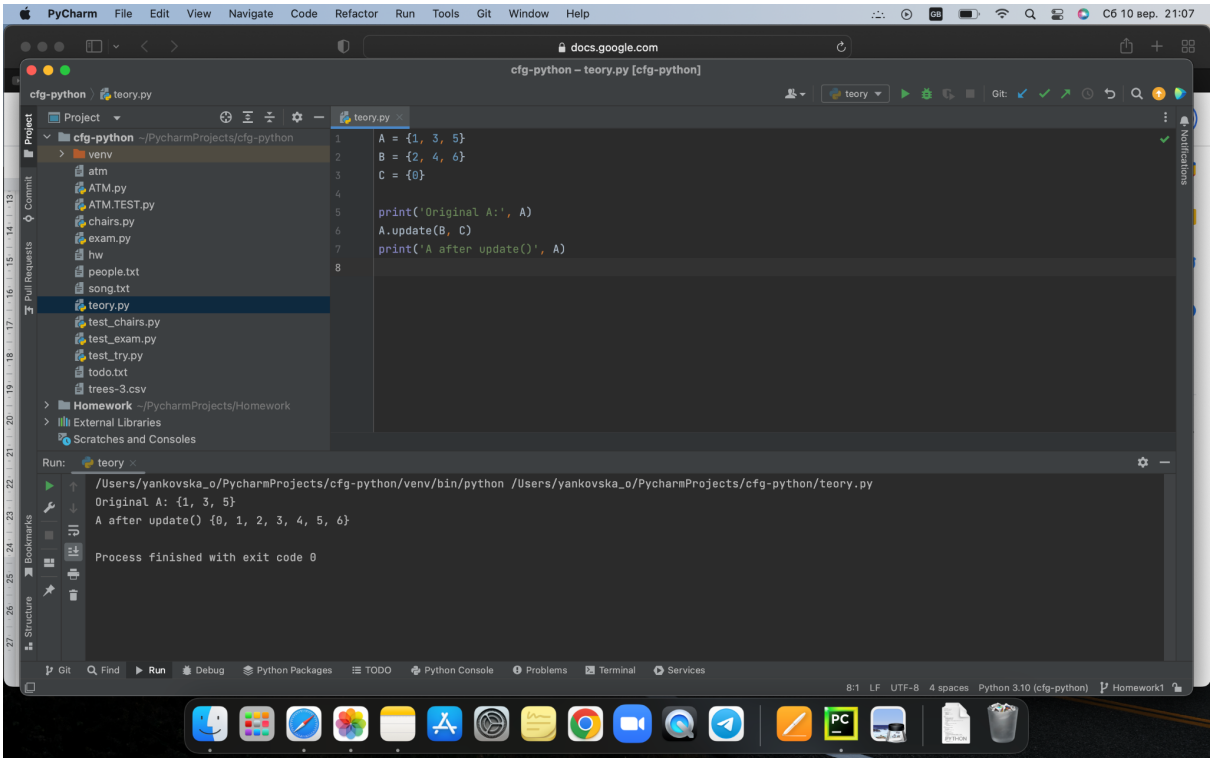
```
A = {'a', 'c', 'd'}
B = {'c', 'd', 2 }
C = {1, 2, 3}

print('A U B =', A| B)
print('B U C =', B | C)
print('A U B U C =', A | B | C)
```



update()

The syntax of the update() method is:
A.update(B)
Here, A is a set and B can be any iterable like list, set, dictionary, string, etc.
The update() method can take any number of arguments. For example,
A.update(B, C, D)
Here,
B, C, D - iterables whose items are added to set A
EXAMPLE:
A = {1, 3, 5}
B = {2, 4, 6}
C = {0}

print('Original A:', A)
A.update(B, C)
print('A after update()', A)

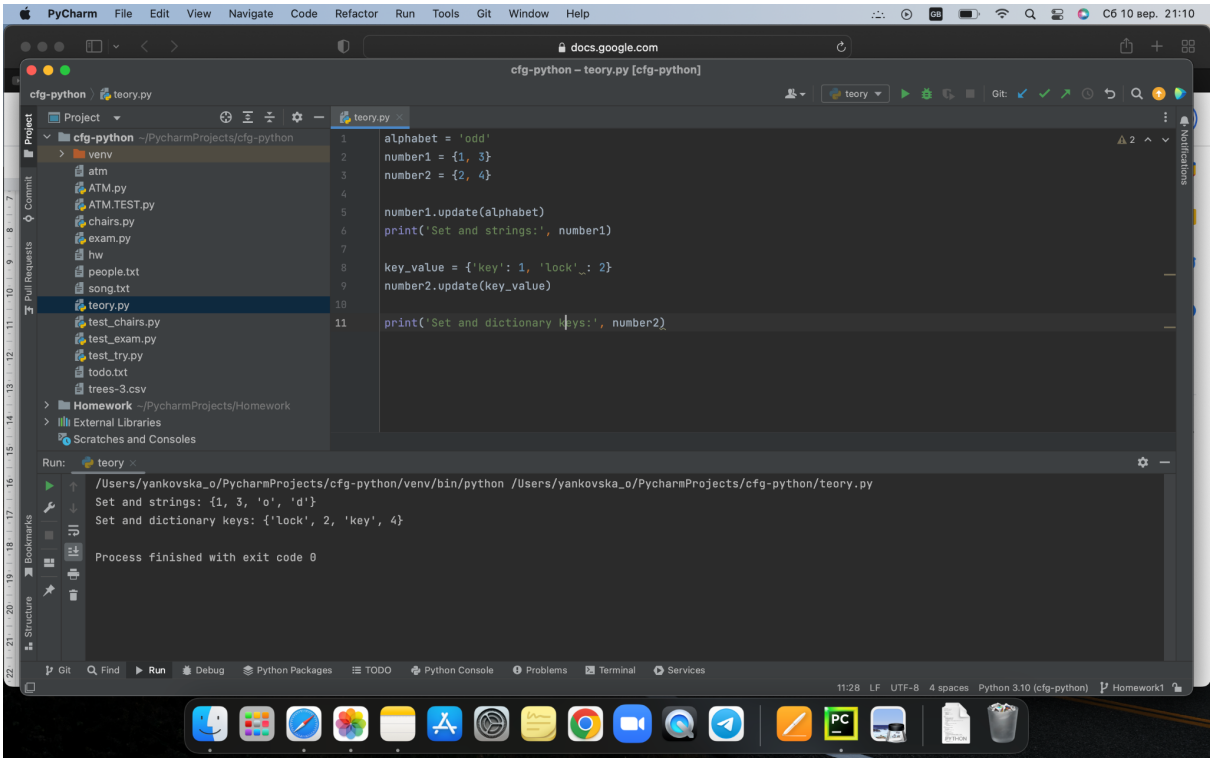


EXAMPLE2:
alphabet = 'odd'
number1 = {1, 3}
number2 = {2, 4}

number1.update(alphabet)
print('Set and strings:', number1)

key_value = {'key': 1, 'lock' : 2}
number2.update(key_value)

print('Set and dictionary keys:', number2)



---

7. Python file methods:

describe each method and provide an example

https://www.w3schools.com/python/python_ref_file.asp

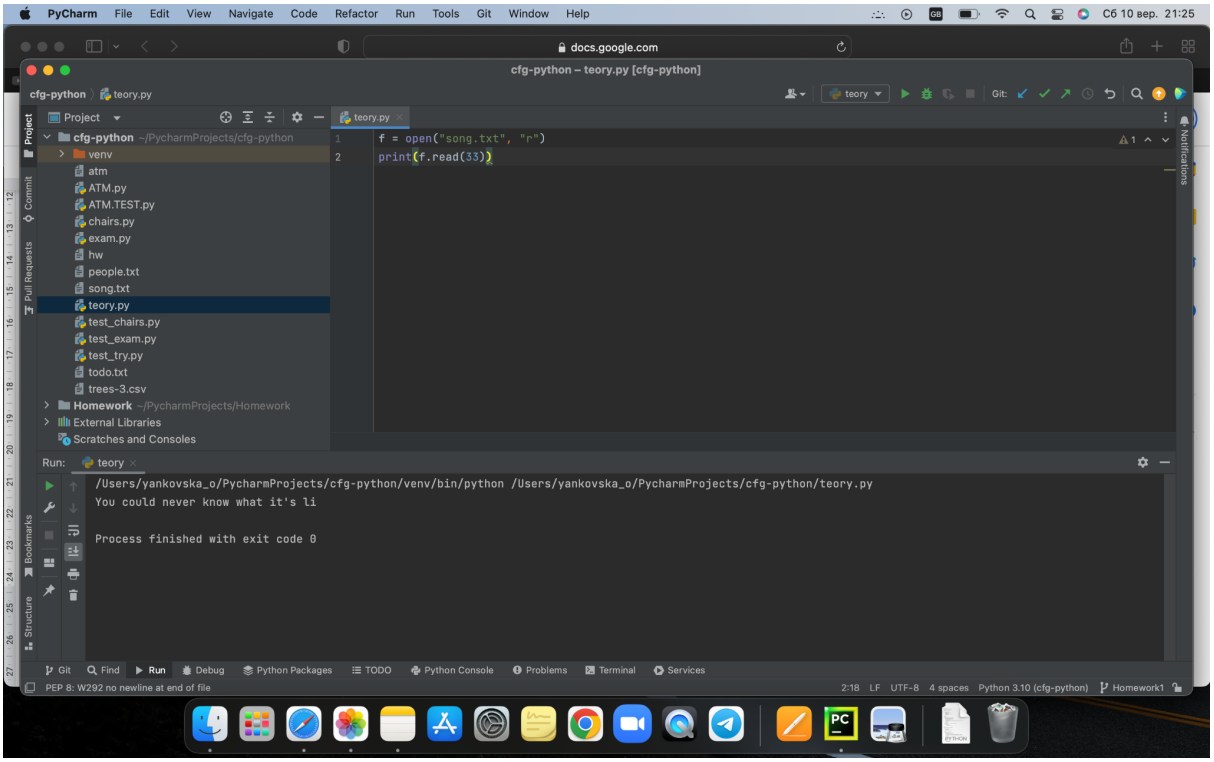
read()

The read() method returns the specified number of bytes from the file. Default is -1 which means the whole file.

Syntax

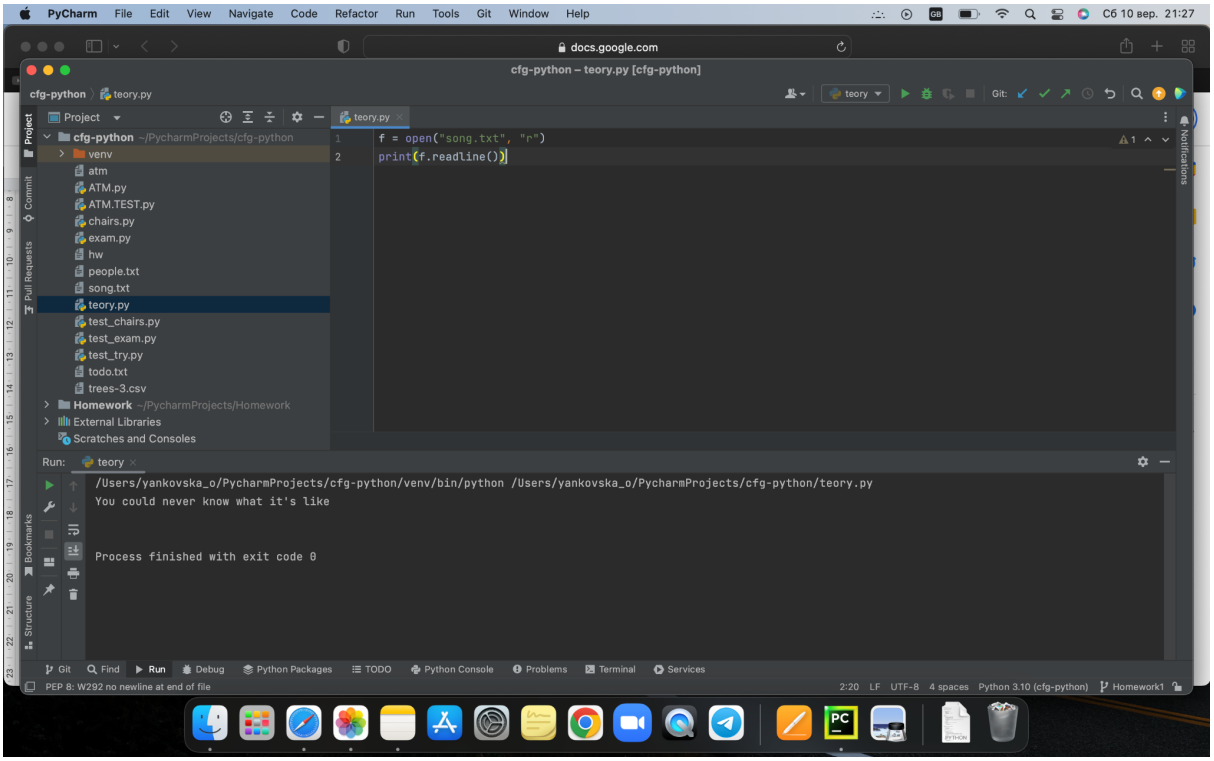
file.read()

```
f = open("song.txt", "r")
print(f.read(33))
```



readline()
The readline() method returns one line from the file.

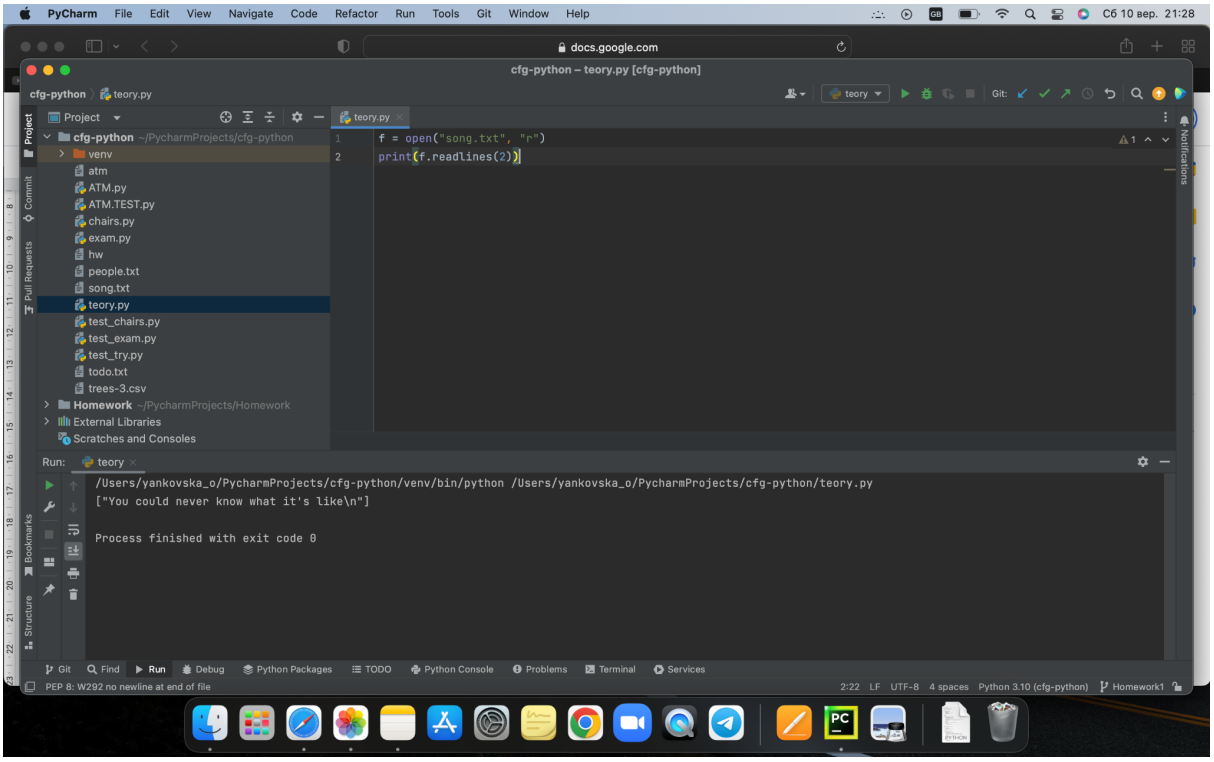
You can also specified how many bytes from the line to return, by using the size parameter.
f = open("song.txt", "r")
print(f.readline())



readlines()

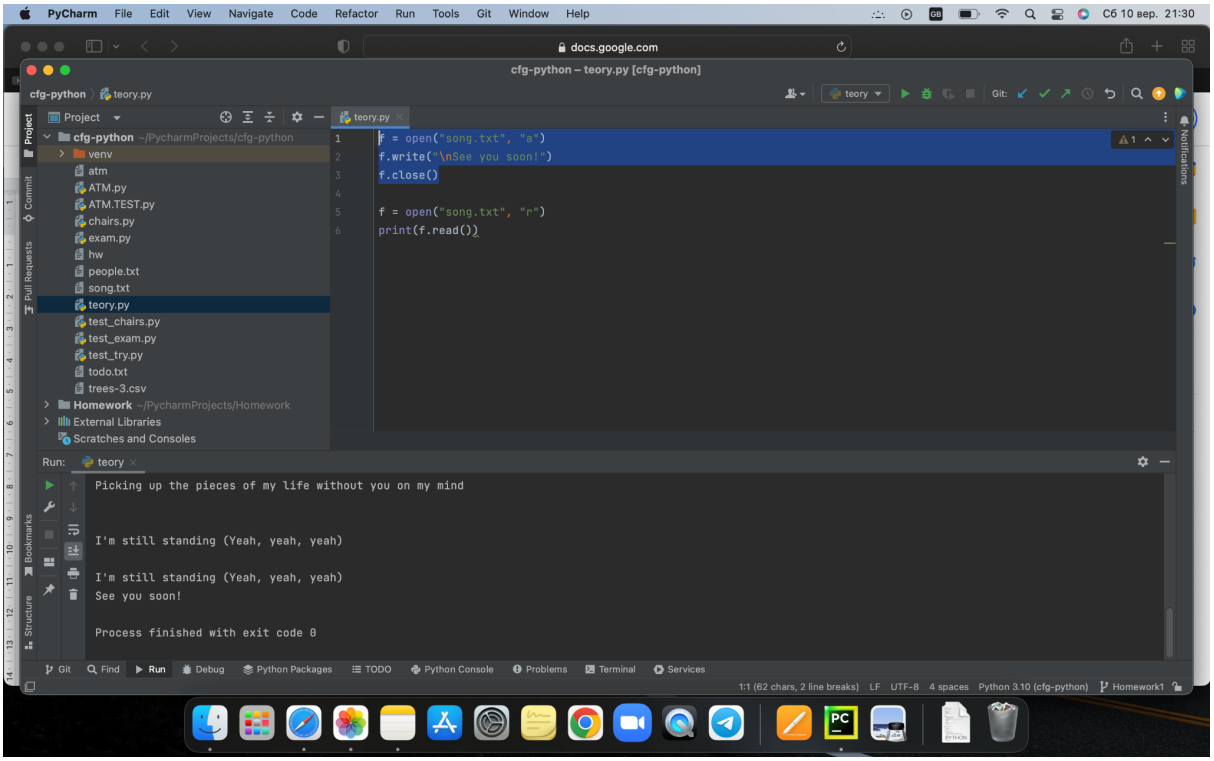
The readlines() method returns a list containing each line in the file as a list item.

```
f = open("song.txt", "r")
print(f.readlines(2))
```



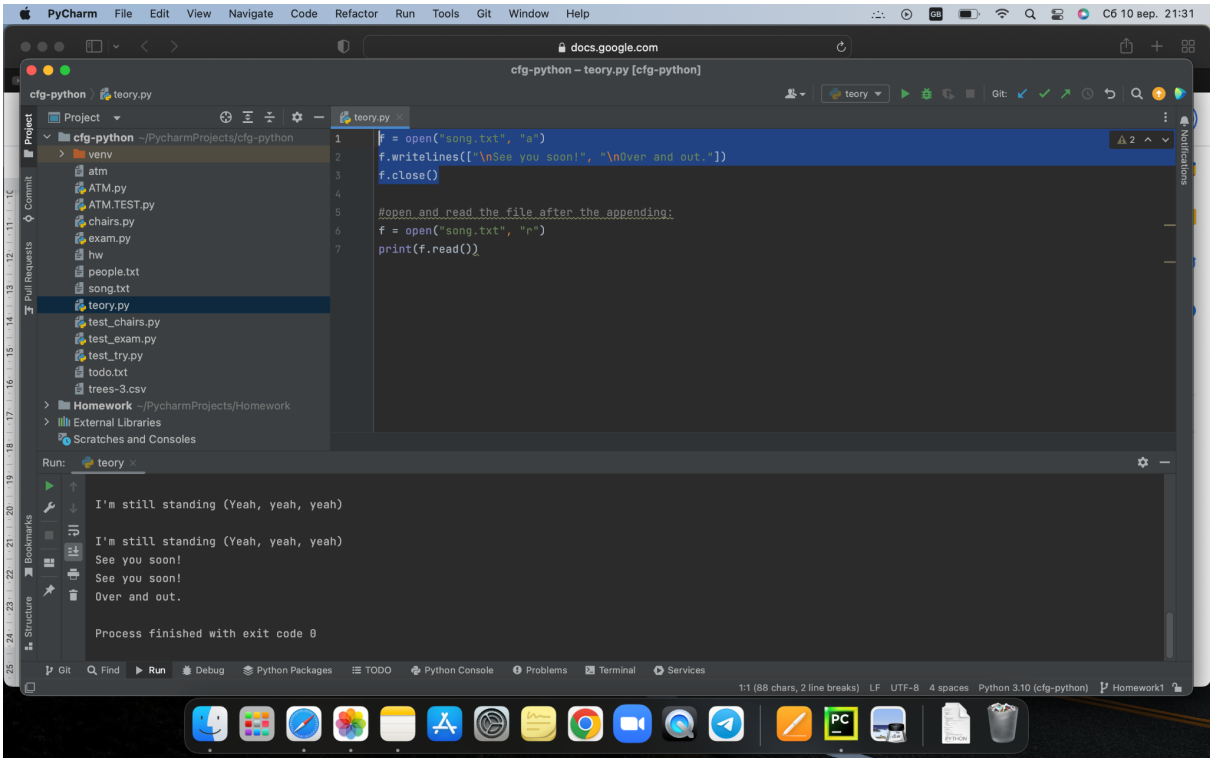
write()
The write() method writes a specified text to the file.

```
f = open("song.txt", "a")
f.write("\nSee you soon!")
f.close()
```



writelines()
The writelines() method writes the items of a list to the file.

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


PyCharm File Edit View Navigate Code Refactor Run Tools Git Window Help
docs.google.com
cfg-python – teory.py [cfg-python]
cfg-python teory.py
Project
cfg-python ~/PycharmProjects/cfg-python
venv
atm
ATM.py
ATM.TEST.py
chairs.py
exam.py
hw
people.txt
song.txt
teory.py
test_chairs.py
test_exam.py
test_try.py
todo.txt
trees-3.csv
Homework ~/PycharmProjects/Homework
External Libraries
Scratches and Consoles
teory.py
f = open("song.txt", "a")
f.writelines(["\nSee you soon!", "\nOver and out."])
f.close()
#open and read the file after the appending:
f = open("song.txt", "r")
print(f.read())
Run: teory
I'm still standing (Yeah, yeah, yeah)
I'm still standing (Yeah, yeah, yeah)
See you soon!
See you soon!
Over and out.
Process finished with exit code 0
Git Find Run Debug Python Packages TODO Python Console Problems Terminal Services
1:1 (88 chars, 2 line breaks) LF UTF-8 4 spaces Python 3.10 (cfg-python) Homework1