

CPSC 231 Assignment 3

Chat231: Creating your own mini-ChatGPT

Weight: 4% of final grade

Due date: Friday November 10, at 11:59pm Mountain Time

Submission: one Python .py file, submit on the D2L Dropbox. You may submit as many times as you wish, and only the latest submission will be graded.

Extensions: You may use your personal days to extend the deadline. An extension request must be submitted using the request form: <https://forms.office.com/r/2wN7KNhYEK>

You have a total of 5 personal days for the entire semester. No penalties for using personal days. All personal day extensions are automatically approved after a request form is submitted. An assignment will not be accepted if it is late with no approved extensions, other than in exceptional circumstances.

Academic Integrity: This work must be completed individually. Please follow academic integrity rules as discussed during lecture. You may discuss your ideas in English (not in Python) with other students as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to never share your code with anyone, except your instructor and TAs.

Notes:

- You should not import anything in your code.
- The existing file server.py imports your code in support_functions. This is how server uses all the code you write in support_functions. You should not change anything in server.py.
- You should not have any code that is outside a function. All your code should be defined inside a function. Your functions are going to be used by server.py.
- For the purpose of the assignment, you are not allowed to use “break”, “continue”, “exit()”, or “quit()” in your code. Instead of stopping your code halfway when encountering some undesirable conditions, learn to adjust your conditions so that your code runs only in the desirable conditions.
- Your code should be written to work with any parameters given by server, not just the specific parameters tested by the autograder.
- You would be assigned a grade of no higher than C if these rules are not followed.

Detailed Descriptions

Think ChatGPT is cool? Well, how about creating your own version of ChatGPT? As a computer science major, you can't just know how to use ChatGPT. You will need to learn how it works behind the scenes so you can **create** your own ChatGPT! That's the future job market – the job of a computer science major is to create ChatGPT and the next awesome tools, and the job of a business major is to use them.

Learning the neural networks that power ChatGPT is way beyond the scope of this course. Let's start with this simpler assignment, and see just how difficult it is to create even a mini version of ChatGPT, called Chat231, that can answer simple questions on CPSC 231.

Step 1:

Create a new py file to work on. Name your file **support_functions.py**. If it is not called this name, rename it.

Step 2:

Download the new file, **server.py**, from the D2L assignment page. `server.py` must be placed in the same directory as your own `support_functions.py` file.

Step 3:

Do not change anything in **server.py**.

Start writing all your code in **support_functions.py**. `server.py` imports `support_functions`, so whatever code you write in `support_functions` will be used by `server`.

Step 4: Your first job is to ensure the files are set up correctly. Run `server.py` (do not run `support_functions.py`). You should see the following:

Starting server on 127.0.0.1:8080.

This Python code is creating a web server on your local computer. The server will keep running until you stop the Python program (or until you crash it with some error in your code).

Now open a web browser (Chrome, Firefox, Edge, etc. but **not** Safari because Apple doesn't like to play with others) and go to

<https://pages.cpsc.ucalgary.ca/~richard.zhao1/chat231/>

You should see:

Chat231 - Your Virtual Assistant for CPSC231

Your name:

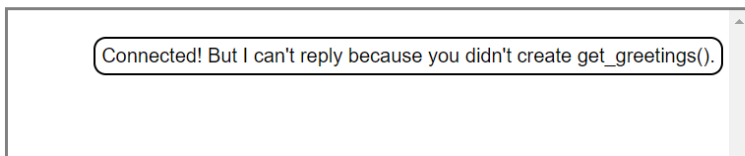


Send a message:

(While keeping your Python code running) Click the button "Connect to server". If you see the following message in your web browser, you've set up correctly.

Chat231 - Your Virtual Assistant for CPSC231

Your name:



Connected! But I can't reply because you didn't create `get_greetings()`.

This web page (called a client) is able to connect with the Python-based server running on your computer. If you don't see the "Connected!" message, your server is not running correctly.

Step 5: Chat231 is complaining that you did not create the function `get_greetings()`, so your task is to create a new custom function in your `support_functions.py` file, called ***get_greetings(name)***. The function has the following:

Parameter: *name*, a string.

Returns: a string.

The function should return the following string:

"Greetings <name>! I am your virtual assistant Chat231."

Here <name> should be whatever the value of the *name* parameter is.

As an example, if the function is called with "John": `get_greetings("John")`

then the returned string should be

"Greetings John! I am your virtual assistant Chat231."

Note that this string is a **return value of the function**. It is not an output, so you should not print it. *This is different than assignments 1 and 2 where you are printing the results to the screen. You are not printing anything here.*

The server in server.py will call your function, get its return value, and then send this return value to the web browser to be displayed.

To test your code, restart the running Python code server.py. **You must restart the server each time you make a change to your Python code.** You don't need to reload the web browser each time you make a change.

Go to the web page, and type in any name, and click "Connect to Server". You should see the following:

Your name:

Greetings Alice! I am your virtual assistant Chat231.

Step 6: Special case. If the function ***get_greetings(name)*** is called with *name* being an empty string "" (because the user did not type in a name), an alternative message should be returned by your function:

"Hello there! I am your virtual assistant Chat231."

Your name:

Hello there! I am your virtual assistant Chat231.

Step 7:

You will create the actual question and answer functionalities of Chat231. But first, you will need to create a helper function that your other function needs to use.

Create a new custom function in your `support_functions.py` file, called ***is_whole_word(word, sentence)***. Create this function from scratch.

The function has the following:

Parameters:

word, a string.

sentence, a string.

Returns: a Boolean

The *Boolean return value* is True if the word exists as a whole word in the sentence. False otherwise.

A whole word is defined to be a word that is not a part of another word. For example:

- In the sentence "oh hi you!", "hi" is a whole word, so the function should return True.
- In the sentence, "high up in the sky", the word "hi" does not exist as a whole word since it is a part of "high", so the function should return False.
- Normally in a sentence, any spaces or punctuations (comma, period, question mark, exclamation mark, colon, and semicolon) separate two words. Numbers do not separate a word. For example, "Chat,231" are two words. "Chat231" is one word.
- However, the string value of the parameter *word* is considered one word, regardless of whether it contains spaces or other symbols in it. For example, if *word* is "final exam", then "final exam" is defined to be one word, and the space in it is considered a part of this word (*this rule overrides the previous rule*).
- If the word does not appear at all in the sentence, the function returns False.

More examples:

Given parameter values when function is called:	Function should return:	Reason
<code>is_whole_word("hi", "oh hi you!")</code>	True	hi is a whole word
<code>is_whole_word("hi", "high up in the sky!")</code>	False	hi is a part of high
<code>is_whole_word("Chat231", "Hi chat231!")</code>	True	Chat231 is a whole word
<code>is_whole_word("Chat,231", "Hi chat,231!")</code>	True	Chat,231 is a whole word since comma is considered a part of this word
<code>is_whole_word("Chat", "Hi chat,231!")</code>	True	Chat is a whole word. The comma separates Chat and 231.
<code>is_whole_word("Chat", "Hi chat231!")</code>	False	Chat is a part of Chat231

This function is case insensitive – meaning any combination of upper/lower case words are accepted as valid, so "HI" counts as appearing in "hi you." *Note: this is a different requirement than assignment 1. You are building a chatbot, after all, and people don't follow proper grammar when chatting.*

You can run the autograder and check to see if it passes its tests.

Step 8:

Create a new custom function in your `support_functions.py` file, called **`get_answers(q_and_a, question)`**. The function has the following:

Parameters:

q_and_a, a dictionary

question, a string.

Returns: a string.

`q_and_a` will be assigned by the caller to be question keywords and their corresponding answers in a dictionary. Here is one example of what it might look like (you can't assume this is the only value for the dictionary, since the dictionary, as a parameter, can be assigned any arbitrary value by whoever is calling this function):

```
{
    "name": "I am Chat231. My nickname is Riley.",
    "textbook": "The textbook we use is called The Python Workbook 2nd Edition.",
    "final exam": "The final exam will be on December 18th at 5-7pm. Check D2L for details."
}
```

The basic idea is:

Your function will scan the user question for any keywords that appear in the dictionary. If a keyword is found, the corresponding answer in the dictionary is returned.

For example, if the user types in "What's your name?" , your function checks this sentence for the keywords. If it sees "name", it assumes the user is asking about name, and returns the answer "I am Chat231. My nickname is Riley."

Now, your **`get_answers`** function must call your own **`is_whole_word`** function at some point to make sure the keyword is, in fact, a whole word and not a part of another word.

For example, if the user types in "What's your namesake?" , your function checks this sentence, calls `is_whole_word` on the keyword "name". The function `is_whole_word` returns False, telling you that "name" is not a whole word in the sentence. You keep checking other keywords. If no other keywords appear in the sentence, Chat231 can't answer the question, and this function should return

"Sorry, I do not understand your question."

See more examples below. Here q_and_a represents the entire dictionary, assuming the dictionary example above is given to the function.

Given parameter values when function is called:	Function should return:
get_answers(q_and_a, "Name!?")	"I am Chat231. ..." (the full sentence)
get_answers(q_and_a, "What Textbook, do we use?")	"The textbook we use is ..."
get_answers(q_and_a, "What's on final exam.")	"The final exam will be on December ..."
get_answers(q_and_a, "Ha ha ha!")	"Sorry, I do not understand your ..."

Everything here is case insensitive – any upper/lower case combination of words should be recognized as words.

You can assume the sentence will contain at most one keyword from the dictionary. If multiple keywords appear in the sentence, choose any one keyword and return its answer.

Chat231 - Your Virtual Assistant for CPSC231

Your name:

The screenshot shows a chat window with a scrollable history. The messages are as follows:

- System message (top): Greetings Cathy! I am your virtual assistant Chat231.
- User message: Final
- System message: Sorry, I do not understand your question.
- User message: Final exam date?
- System message: The final exam will be on December 18th at 5-7pm. Check D2L for details.

Test cases

Your code will be tested on some test cases using an AI auto-grader. The result will determine your grade on the assignment. To help you fix your problems, the auto-grader is provided to you, so you can test your code yourself before you submit. Be sure to do this to make sure your code runs correctly and there are no typos.

To test your code on the test cases:

Step 1: Your Python file needs to be on a lab machine. The auto-grader only runs on a Linux machine.

If your Python file is on your own computer, you will need to copy your Python file to a Linux machine in the CPSC labs.

- If you are sitting in front of a lab machine, you could use a USB stick/flash drive to copy the file over.
- If you are at home, you will need to remotely transfer your file to a CPSC server machine. You could use scp. ([SCP instructions](#))

Step 2: If you are sitting in front of a lab machine, you can open a terminal. If you are not, you will need to use ssh to connect to a CPSC server machine. ([SSH instructions](#))

Step 3: In your Terminal or Bash window, run the auto-grader by typing

```
/home/profs/richard.zhao1/231/a3/autograde support_functions.py
```

assuming the file is in your current directory.

You should see the results of the auto-grader.

You should also test your code without using the auto-grader.

Once you are ready to submit, submit your code on D2L. The code you submit on D2L is the one your TA will grade.

Grading

Grade Point	Letter Grade	Guidelines subject to notes stated above
4	A+	Fulfill all assignment specs
4	A	One failed test
3.7	A-	Two failed tests
3.3	B+	Three failed tests
3	B	Four failed tests
2.7	B-	Five failed tests
2.3	C+	Six failed tests
2	C	Seven failed tests
1.7	C-	Eight failed tests
1.3	D+	Nine failed tests
1	D	More than nine failed tests, or code does not run due to syntax errors
0	F	Barely started code, or no submission, or late submission with no extension approved

Your submitted code should be clearly documented with comments. Comments need to include: your name, descriptions of what your code does, and citing any sources you have used to complete this assignment. Code without proper comments could receive a letter grade reduction.

The assignment will be graded out of 4, with the grade based on the code's level of functionality and conformance to the specifications.