

## Problem 1.

Question 10 from the 2016 exam is really generate-all-combinations question in disguise.

Recall that we wrote the code to generate all permutations of an alphabet as follows:

```
def all_combinations(n, alphabet, start_str = ''):
    if n == 0:
        return [start_str]
    else:
        res = []
        for letter in alphabet:
            res += all_combinations(n-1, alphabet, start_str + letter)
        return res
```

Suppose that, now, instead of generating all combinations of characters, you want to generate all combinations of list elements. For example, if the list of elements is [1, 2, 3] and  $n = 2$ , you want to generate the list [[1, 1], [1, 2], [1, 3], [2, 1], [2, 2], [2, 3], [3, 1], [3, 2], [3, 3]].

Change the code from lecture to do this. Warning: you should not try to have a default argument of `start_list = []`, as this will not work as you expect (try it!). Just use `start_list = None` and then inside the function check if `start_list` is `None`, and if so, set it to `[]`.

Now, you are ready for Q10 from 2016: generate a list of all potential cliques, check for each whether it actually is a clique, and return the largest one.

**Suggestion:** just generate all possible lists of  $n$  friends, with repetitions. To do that, just change string operations to list operations. After that, check if each `set(potential_clique)` is an actual clique.

## Problem 2.

Recall that you can use the `exec` function to execute a string as Python code. For example:

```
code_str = 'print("Hello, World!")'
exec(code_str)
```

Write a function that generates and then executes the code for a function that takes in  $n$  integers and returns their sum. For example, the end result should be:

```
# sum5(5, 6, 7, 8, 9) # ERROR: sum5 is not defined
```

```
def make_sum_fn(n):
    code_str = 'def sum' + str(n) + '('
    #....
    exec(code_str, globals())
```

```
make_sum_fn(5)
print(sum5(5, 6, 7, 8, 9)) # prints 35 since sum5 is now defined!
```

You must use the `exec` strategy. (Note: you can also do this with `*args`, but that is not the point of this exercise.)

## Setup

Create a folder for the work in this lab named `lab11`.

Download the “notopenai client” from

<https://www.cs.toronto.edu/~guerzhoy/190/notopenai.zip>

Unzip the folder and put it in the `lab11` folder.

Create `lab11.py` and put it in the `lab11` folder as well. Open `lab11.py`.

Obtain your OpenAI API key from <https://esc180-lab.vercel.app/>

## Problem 3.

You can now interact with the OpenAI GPT-3.5-Turbo model using the ESC180 client.

```
from notopenai import NotOpenAI
import json
```

```
API_KEY = # PASTE YOUR API KEY HERE AS A STRING
```

```
CLIENT = NotOpenAI(api_key=API_KEY)
```

```
def get_response(prompt):
    chat_completion = CLIENT.chat.completions.create(
        messages=[
            {
                "role": "user",
                "content": prompt,
            }
        ],
        model="gpt-3.5-turbo", # the GPT model to use
    )
    response_str = chat_completion.choices[0].message.content
    return response_str
```

```
get_response("What is the meaning of life?")
```

## Problem 4.

Write a function to interact with the OpenAI GPT-3.5-Turbo model.

To do this, you must append the conversation history to the prompt you send. For example, the first time you send a prompt, you just send in the string "What is the mening of life?". The second time, you send the entire chat history:

```
'''Q: What is the mening of life ? A: [Whatever GPT said]
Q: "OK but what is the meaning of Praxis specifically?'''
```

Write a while-loop that repeatedly prompts GPT for answer while sending the chat history to it every time.

## VIBE CODING

*Vibe coding*, according to Merriam-Webster, is a recently-coined term for the practice of writing code, making web pages, or creating apps, by just telling an AI program what you want, and letting it create the product for you. In vibe coding the coder does not need to understand how or why the code works, and often will have to accept that a certain number of bugs and glitches will be present. The verb form of the word is ‘vibe code’. The term was coined by Andrej Karpathy, a UofT CS 2009 graduate.

You will learn how to use the OpenAI application programming interface (API), which allows applications to communicate with each other by sending and receiving data, to generate Python function code, execute the generated code, and test its correctness. You will analyze when the large language model (LLM) generates accurate code and when it fails.

### Problem 5. Prompt Engineering

Write a prompt that asks the GPT-3.5-turbo to generate a Python function for you that outputs the factorial of the input  $n$ . Extract the code portion from the response and print out the resulting code.

### Problem 6. Test the Code Generated by AI

Given a few test cases in the following format:

```
test_cases = [
    {"input": 3, "expected_output": 6},
    {"input": 4, "expected_output": 24},
]
```

Write the function `check_result(generated_code, test_cases)`, and print out how many cases passed and failed.

You can use `exec` to define and use a function contained in a string:

```
def check_result(generated_code, test_cases):
    """
    Check which test cases failed from the generated code
    generated_code - str
    test_cases - dict
    e.g., if the generated_code is:
    def fun(x):
        return x
    """
    exec(generated_code, globals())
    # Execute the string that
    # contains the function will make
    # the function available to be called below
    # print(fun(1)) # this will print 1 in the example above.
```

In your prompt, you might want to explicitly ask for the generated function to be named, *e.g.*, `fun(x)`, and you might ask to generate the function only.

Note that executing code without looking might be a security risk! You might like to print the code and then ask the user (yourself) to agree to execute it.

## Problem 7. Test Cases

Here is a task on which GPT-3.5 usually doesn't succeed:

```
s = '''Date,Character,Age,HeightCm,AppleCount,MoodRating
2025-01-15,Snow White,14,157.5,1,8.5
Doc,200,91.4,3,7.2
2025-01-16,Grumpy,199,89.0,0,3.4
2025-01-16,202,94.0,2,9.7
2025-01-17,Sleepy,202,90.2,1,6.3
Bashful,198,88.5,1,5.8
2025-01-18,Sneezy,197,92.3,2,7.4
2025-01-18,Dopey,195,87.1,4,8.9
2025-01-19,,42,175.6,0,2.1
Prince,25,185.3,2,9.5
2025-01-20,Huntsman,38,178.4,1,6.7
2025-01-20,250,92.0,3,7.3
2025-01-21,Forest Animals,5,30.5,4,9.2'''

print(get_response('''Write Python code to parse a CSV string
formatted like the following. Result needs
to be a dictionary of dictionaries\n\n\n'' + s))
```

Note that here, some dates and character names are missing, and GPT-3.5 is not (usually) smart enough to figure out how to handle that. Write test cases for the function, experiment with different prompts on ChatGPT.com, and demonstrate how you can generate code in our framework that passes the test cases.

Note: this is not a trivial task. Make sure that the character names are only names and never dates, *etc*