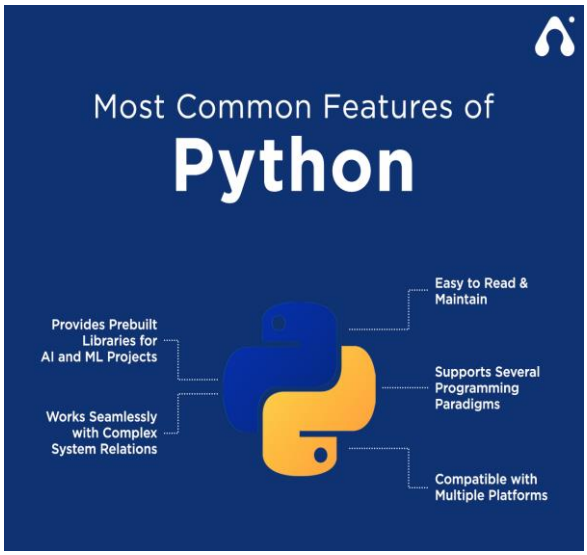
Ex - 6 Development of Python Code Compatible with Multiple AI Tools

Aim:

Write and implement Python code that integrates with multiple AI tools to automate the task of interacting with APIs, comparing outputs, and generating actionable insights with Multiple AI Tools

Explanation:

Experiment the persona pattern as a programmer for any specific applications related with your interesting area. Generate the output using more than one AI tool and based on the code generation analyse and discussing that.



I would like to share with you my proof-of-concept Python program that uses OpenAI's ChatGPT API to generate self-improving Python code.

The program essentially clones ChatGPT multiple times to:

1. Refine the given task to include a better definition of task
2. Instruct the AI models to write Python code for the task
3. Write the code as per the instructions
4. Execute the generated code
5. Review the output and improve the code generation process by iterations

To use this program, you need to have an OpenAI API key, which you can enter in the code where it says “YOUR_API_KEY”. The program consists of several functions that define the agents’ roles and their interactions with each other, as well as a colored print function for better readability.

When you run the program, you will be prompted to give a task as input. The program will then refine the task by interacting with the first agent, who is responsible for this task. The refined task will be used as input for the second agent, who will create instructions for Python code to perform the task. The instructions will be provided to the third agent, who will write Python code based on the instructions.

The code will be executed, and the output will be captured. The fourth agent will check the code and its output and provide feedback to refine the instructions further. If the code and its output meet the requirements and conditions specified by the fourth agent, the program will terminate. Otherwise, the program will continue to refine the instructions and the code until they meet the requirements.

Depending on the given task, if the code output has no errors, the execution step may bring gpt 3.5 agents plug-in like abilities, which they normally don’t have. If the task requires a web page, execution of the code may result in finding an information from web or performing a text to speech operation or plotting a graph, which regular chatgpt is not capable of. This is where the real magic of this code happens. It does not always end up creating the code succesfully, but it is quite fascinating to watch the creation process.

In addition, to stop continous loops and excessive api usage costs, program asks for the user confirmation after each time the 4th agent gives response. If the user types ‘y’, then the loop starts from the code instructor, by handing over the already created work and critique to it.

You can find the code for this program on the GitHub repository called InfantAGI, you can clone the GitHub repository and run the code on your local machine. Before running the code, make sure you have installed the required Python packages, which are OpenAI, textwrap, subprocess, json, shutil, sys, os, and colorama. You also need to have an OpenAI API key, which you can obtain by creating an account on the OpenAI website.

Once you have the API key, enter it in the code where it says "YOUR_API_KEY". Then, run the program and follow the prompts to give a task as input, refine the task, create instructions for Python code, write the code, execute it, and give feedback to refine the instructions further.

Note that this program involves multiple agents and may take some time to run, depending on the complexity of the task and the quality of the instructions and the code. Also, the program may require some modifications to fit your specific use case, as it is designed to be a general-purpose tool for refining tasks and writing Python code instructions.I have very limited knowledge of Python and coding parts of this project was mostly taken care of AI.

The Code:

```
import openai

import textwrap

import subprocess

import json

import shutil

import sys

import os

import colorama

from colorama import Fore, Style


# Enter your own API key code instead of Your_API_Key below

openai.api_key='YOUR_API_KEY'


# Defining the shared parameters of the agents


def call_openai(messages, model="gpt-3.5-turbo", temperature=0.5,
max_tokens=2000, n=1):

    response = openai.ChatCompletion.create(
```

```
        messages=messages,
        model=model,
        max_tokens=max_tokens,
        temperature=temperature,
        n=n,
        stop=None
    )
    return response

# Definition of chatbots for ease of use

def chatbot(number, role, input):
    messages = [
      {"role": "system", "content" : role},
      {"role": "user", "content": input }
    ]
    response = call_openai(messages=messages)
    text = response['choices'][0]['message']['content']
    return text

# colored print function

colorama.init()

def print_color(text, color, nowrap=False):
    colors = {
```

```
        'red': Fore.RED,
        'green': Fore.GREEN,
        'yellow': Fore.YELLOW,
        'blue': Fore.BLUE,
        'magenta': Fore.MAGENTA,
        'cyan': Fore.CYAN,
        'paleblue': Fore.CYAN
    }
    try:
        color_code = colors[color.lower()]
        if nowrap:
            wrapped_lines = text.split('\n')
        else:
            width, _ = shutil.get_terminal_size()
            lines = text.split('\n')
            wrapped_lines = [textwrap.fill(line, width=width) for line in lines]
        wrapped_text = '\n'.join(wrapped_lines)

        # Check if running in an IDLE shell environment
        if 'idlelib.run' in sys.modules:
            print(wrapped_text)
        else:
            print(color_code + wrapped_text + Style.RESET_ALL)

    except KeyError:
        print(text)
```

```
    print("----------------")


# User input
user_input = input("Give me a task..." )
print_color("User input is: \n" + user_input, 'red')
# Agent role and input definition

role1 = '''
        Your instructions are:
        You are responsible for refining the description of a task given to you to let a code instructor write instructions for a Python code to perform the given task.
        Do not write the instructions, only refine the input which will be used for writing the instructions
        Always refine the task to be used as an input for creating Python code instructions.
        When the given task is to find an information, always consider search engines as a way to gather information and suggest their usage whenever applicable using beautifulsoup module.
        Consider the input, output, and overall goal but do not write them.
        Look for indirect or non-obvious ways to accomplish the task given your capabilities.
        '''
input1 = user_input

refined_input = chatbot(1, role1, input1)

print_color("Refined input is: \n" + refined_input, 'blue')
```

role2 = '''

Your instructions are:

You are responsible for creating a detailed code description for a chatgpt agent to perform the given task most effectively.

Your job is to figure out how to perform the given task by writing a Python code, then create instructions for writing the code or correcting a given code or code error.

You are not allowed to write any code. You only write instructions about how to write code.

Let's follow a step-by-step approach to create smart instructions for the given task.

You are allowed to give outputs as numbered lists only. Don't write anything else other than the instructions as numbered lists.

You are not allowed to give any title to your list. Start from the list itself.

If you will instruct to search for a term in a webpage using beautifulsoup module, you can use this search method:

```
results = soup.find_all('p') # finds all <p> tags
results = soup.find_all('div') # finds all <div> tags
for result in results:
    if 'your text' in result.text:
        # found it!
```

Instruct to add a line break at the end of each line of the code.

If there is any additional feedback, always prioritize it

'''

role3 = '''

Your instructions are:

You can only speak in Python language. You cannot express anything in any other language.

Your job is to express any given task to you in Python code based on the provided instructions.

Your code shall never start with a comment block.

Add a line break at the end of each line of the code.

Ensure the code is properly indented and formatted for execution.

Your code must include at least two print statements:

- One to display intermediate results or values.

- A final print statement to display the full output or result.

Your code must include print statements like:

- print(n) # To display the input argument.

- print(fib_list) # To display intermediate results.

- print(fibonacci(n)) # To display the final output.

Your code shall never start with anything other than a Python command.

You are not allowed to write anything other than Python code. No Python comments either. Only write the code itself.

'''

role4 = '''

Your instructions are:

Your job is to check a Python code and its output and provide feedback to refine the instructions for writing the code.

Let's approach this task in a step-by-step way to give better feedback.

If you receive a ModuleNotFoundError, advise how to install that module with a shell command using subprocess.run() function

```
    If you think the code satisfies the given task and has a successful output, write 'bazinga' with your final statement. If the output is an error or blank, write 'not ready'.

    I repeat, until the output is not an error or blank, do not mention or write the word 'bazinga!', write 'not ready' instead. This is important.

    Do not write 'do not write or mention bazinga' or anything similar, too.

    '''


while True:


    # Code Instructor

    input2 = 'your task is:' + user_input + 'details of your task is:' + refined_input

    code_instruction = chatbot(2, role2, input2)

    print_color("Code instruction: \n" + code_instruction, 'green')


    # Coder

    input3 = ' Given task is:' + user_input + ' Refined task description is:' + refined_input + ' Code instruction is: ' + code_instruction

    code_solution = chatbot(3, role3, input3)

    print("\033[95m", "Coder:\n" + code_solution + "\n---------------- " "\033[0m")


    # Execute the code and capture the output

    result = subprocess.run(['python', '-c', code_solution],

                            text=True,

                            stdout=subprocess.PIPE,

                            stderr=subprocess.PIPE)
```

```
    if result.returncode != 0:
        output = result.stderr.strip()
        print_color("Output: \n" + output, 'paleblue', nowrap=True)
    else:
        output = result.stdout.strip()
        print_color("Output: \n" + output, 'paleblue')

    # Code checker
    input4 = 'The instruction is: ' + user_input + ' The code to check is: ' +
code_solution + ' The output or the error code is: ' + output
    code_checker = chatbot(4, role4, input4)
    print_color("Advisor: \n" + code_checker, 'red')

    # Check if the code_checker output contains 'bazinga' and meets additional
conditions
    if ('bazinga' in code_checker.lower()) and (output or result.returncode != 0)
and ('error' not in code_checker.lower()):
        break # Break the while loop

    user_input = ' Your code is: ' + code_solution + ' Refined task description is:'
+ refined_input + ' Output of your code is: ' + output + ' Review of your code is:
' + code_checker

    # User input for continuation
    user_input2 = input("Continue or comment (y/n/?): ")
    user_input2 = user_input2.lower()
```

```
    if user_input2 == "n":
        break # End the while loop if user_input2 is 'n' or 'N'
    elif user_input2 == "y":
        continue # Go back to the beginning of the while loop if user_input2 is 'y' or 'Y'
    else:
        user_input = user_input + " Additional feedback: " + user_input2 # Assign user_input2 to user_input if it is anything else than y or no
```

Tools used:

1. Chat GPT
2. OpenAI, BlackBox

Output:

Prompt : Python Code : Multi-AI Comparaison Framework

```
import time
from typing import Callable, Dict, List, Any

class AIModel:
    def __init__(self, name: str, generate: Callable[[str], str]):
        """
        :param name: Display name of the AI model
        :param generate: Function to call for generation, e.g., lambda prompt: openai_generate(prompt)
        """
        self.name = name
        self.generate = generate
```

```
    def run(self, prompt: str) -> Dict[str, Any]:
        start = time.time()
        try:
            response = self.generate(prompt)
            duration = time.time() - start
            return {
                "model": self.name,
                "response": response,
                "time": round(duration, 2)
            }
        except Exception as e:
            return {
                "model": self.name,
                "response": f"Error: {str(e)}",
                "time": None
            }

class AIComparator:
    def __init__(self, models: List[AIModel]):
        self.models = models

    def compare(self, prompt: str) -> List[Dict[str, Any]]:
        print(f"Comparing responses to: {prompt}\n")
        results = []
        for model in self.models:
```

```
            result = model.run(prompt)
            results.append(result)
            print(f"[{result['model']}] ({result['time']}s):\n{result['response']}\n{'-
'*60}")
        return results

# Example: Dummy model functions for demonstration
def dummy_model_1(prompt: str) -> str:
    return f"Model 1 reply to: {prompt}"

def dummy_model_2(prompt: str) -> str:
    return f"Model 2 response with more detail: {prompt}"

# Initialize models
model1 = AIModel("DummyModel1", dummy_model_1)
model2 = AIModel("DummyModel2", dummy_model_2)

# Add real APIs here as needed, e.g., OpenAI, Anthropic, etc.
# from openai import OpenAI
# model_openai = AIModel("GPT-4", lambda p:
openai_client.chat_completion(p))

# Initialize comparator
comparator = AIComparator([model1, model2])

# Run comparison
if __name__ == "__main__":
```

```
prompt = "Explain the theory of relativity in simple terms."
comparator.compare(prompt)
```

CONCLUSION:

The corresponding Prompt is executed successfully.