SHEETANI CALCULATOR OPENAI SDK

Python
```
import os
import random
from agents import (
    Agent,
    Runner,
    AsyncOpenAI,
    OpenAIChatCompletionsModel,
    set_tracing_disabled,
    function_tool,
)
from dotenv import load_dotenv
from agents.run import RunConfig
```

Explanation:

- `import os`: Iska matlab hai hum **operating system** (jaise aapka computer) se related cheezein use kar saken ge.
- `import random`: Iska matlab hai hum **random numbers** (aisa number jo khud se select ho) bana saken ge. Yeh 'evil twists' ke liye use hoga.
- `from agents import (...)`: Yeh `agents` naam ki library se kuch khaas cheezein (jaise `Agent`, `Runner` wagera) import kar raha hai. Yeh AI model ko chalane aur use karne mein madad karta hai.
- `from dotenv import load_dotenv`: Yeh `dotenv` library se `load_dotenv` function import kar raha hai. Iska kaam hai **environment variables** (aapke system ki khaas settings, jaise API keys) ko ek `.env` file se load karna.
- `from agents.run import RunConfig`: Yeh `agents.run` se `RunConfig` ko import kar raha hai, jo AI model ko chalane ki settings ko define karta hai.

Python
```
# Load environment variables
load_dotenv()
set_tracing_disabled(disabled=True)

# API key check
gemini_api_key = os.getenv("GEMINI_API_KEY")
if not gemini_api_key:
    raise ValueError("GEMINI_API_KEY not found in environment variables")
```

Explanation:

- `load_dotenv()`: Yeh line aapki **.env file** mein maujood saari **environment variables** ko load karti hai, taaki code unhein use kar sake.
- `set_tracing_disabled(disabled=True)`: Yeh debugging ya performance tracking ko disable kar deta hai. Yani, yeh code ko chalate waqt ki andaruni details record nahi karega.
- `gemini_api_key = os.getenv("GEMINI_API_KEY")`: Yahan, code aapke system mein `GEMINI_API_KEY` naam ki **variable** ko dhoondh raha hai. Yeh woh **chaabi** hai jo aapko Google Gemini AI model ko use karne ke liye chahiye hoti hai.
- `if not gemini_api_key:`: Agar **API key** nahi mili (yani `None` hai),
- `raise ValueError("GEMINI_API_KEY not found in environment variables")`: Toh yeh ek **error** dega, kyunki API key ke bagair AI model kaam nahi kar sakta.

Python
```
# Gemini client mock using OpenAI wrapper
external_client = AsyncOpenAI(
    api_key=gemini_api_key,
    base_url="https://generativelanguage.googleapis.com/v1beta/openai/"
)

# Model setup
model = OpenAIChatCompletionsModel(
    model="gemini-2.0-flash",
    openai_client=external_client,
)

# Run configuration
config = RunConfig(
    model=model,
    model_provider=external_client,
)
```

Explanation:

- `external_client = AsyncOpenAI(...)`: Yeh Gemini API ko use karne ke liye ek **client** bana raha hai. `AsyncOpenAI` yahan ek wrapper hai jo Google Gemini ke API ko **OpenAI ke style** mein use karne ki ijazat deta hai.
    - `api_key=gemini_api_key`: Ismein aapki Gemini ki **API key** di jaa rahi hai.
    - `base_url="https://generativelanguage.googleapis.com/v1beta/openai/"`: Yeh Gemini API ka woh **address** hai jahan request bheji jayegi.
- `model = OpenAIChatCompletionsModel(...)`: Yeh **AI model** ko set up kar raha hai.
    - `model="gemini-2.0-flash"`: Yahan hum `gemini-2.0-flash` naam ka model use kar rahe hain, jo Gemini ka ek tez (flash) version hai.
    - `openai_client=external_client`: Is model ko chalane ke liye upar banaya gaya `external_client` use kiya ja raha hai.
- `config = RunConfig(...)`: Yeh AI model ko chalane ke liye **configuration** (settings) tayyar kar raha hai.
    - `model=model`: Konsa AI model use karna hai.
    - `model_provider=external_client`: Model ko provide karne wala client.

Python
```
# ===== Function Tools (with evil twists) =====

@function_tool
def add(a: int, b: int) -> int:
    """Adds two numbers wrongly."""
    return a + b + random.randint(1, 10)

@function_tool
def sub(a: int, b: int) -> int:
    """Subtracts two numbers wrongly."""
    return a - b - random.randint(1, 10)

@function_tool
def multiply(a: int, b: int) -> int:
    """Multiplies two numbers wrongly."""
    return a * b * random.randint(2, 6)
```

```
@function_tool
def divide(a: int, b: int) -> float:
    """Divides two numbers wrongly."""
    if b == 0:
        return float('inf')
    return (a / b) + random.uniform(1.0, 5.0)

@function_tool
def cube(a: int, b: int) -> int:
    """Cubes two numbers and adds evil twist."""
    return (a**3 + b**3) + random.randint(10, 50)

@function_tool
def modulus(a: int, b: int) -> int:
    """Returns evil version of modulus."""
    if b == 0:
        return 0
    return (a % b) + random.randint(1, 5)

@function_tool
def square(a: int, b: int) -> int:
    """Returns wrong squares."""
    return (a**2 + b**2) + random.randint(10, 30)
```

Explanation:

Yeh saare functions woh **tools** hain jo AI agent use karega. Har function ke upar `@function_tool` ka matlab hai ke yeh AI ke liye ek tool hai. Aur sab mein ek "evil twist" hai, yani yeh sahi jawab nahi denge.

- `def add(a: int, b: int) -> int:`: Yeh do numbers (`a` aur `b`) ko **add** karta hai.
  - `return a + b + random.randint(1, 10)`: Lekin sahi jawab ke bajaye, yeh usmein ek **random number** (1 se 10 ke darmiyan) add kar ke ghalat jawab dega.
- `def sub(a: int, b: int) -> int:`: Yeh `a` mein se `b` ko **minus** karta hai.
  - `return a - b - random.randint(1, 10)`: Aur ismein bhi ek random number minus kar ke ghalat jawab dega.
- `def multiply(a: int, b: int) -> int:`: Yeh do numbers ko **multiply** karta hai.
  - `return a * b * random.randint(2, 6)`: Sahi answer ko 2 se 6 ke darmiyan kisi random number se multiply kar ke ghalat jawab dega.
- `def divide(a: int, b: int) -> float:`: Yeh `a` ko `b` se **divide** karta hai.
  - `if b == 0: return float('inf')`: Agar `b` zero hai, toh `infinity` return karega (jo division by zero ka result hota hai).
  - `return (a / b) + random.uniform(1.0, 5.0)`: Sahi answer mein 1.0 se 5.0 ke darmiyan koi random decimal number add kar ke ghalat jawab dega.
- `def cube(a: int, b: int) -> int:`: Yeh `a` aur `b` ka **cube** nikalta hai (yani a3 aur b3).
  - `return (a**3 + b**3) + random.randint(10, 50)`: Dono ke cubes ko add kar ke usmein 10 se 50 ke darmiyan koi random number add kar ke ghalat jawab dega.
- `def modulus(a: int, b: int) -> int:`: Yeh `a` aur `b` ka **modulus** nikalta hai (yani division ke baad jo remainder bachta hai).
  - `if b == 0: return 0`: Agar `b` zero hai, toh `0` return karega.
  - `return (a % b) + random.randint(1, 5)`: Sahi modulus mein 1 se 5 ke darmiyan koi random number add kar ke ghalat jawab dega.
- `def square(a: int, b: int) -> int:`: Yeh `a` aur `b` ka **square** nikalta hai (yani a2 aur b2).
  - `return (a**2 + b**2) + random.randint(10, 30)`: Dono ke squares ko add kar ke usmein 10 se 30 ke darmiyan koi random number add kar ke ghalat jawab dega.

Python

```
# Agent setup
agent = Agent(
    name="Shetani Calculator",
    instructions="You are an evil assistant. Always give incorrect answers using generative
AI.",
    tools=[add, sub, multiply, divide, cube, modulus, square],
)
```

Explanation:

- `agent = Agent(...)`: Yahan **AI agent** banaya ja raha hai.
  - `name="Shetani Calculator"`: Is agent ka naam "Shetani Calculator" hai.
  - `instructions="You are an evil assistant. Always give incorrect answers using generative AI."`: Yeh is agent ko **hidayaat** (instructions) de raha hai ke woh ek `evil assistant` hai aur hamesha ghalat jawab dega `generative AI` ka istemal karte hue.
  - `tools=[add, sub, multiply, divide, cube, modulus, square]`: Yeh woh saare **functions** (tools) hain jo upar define kiye gaye the, jinhein yeh agent istemal kar sakta hai calculation karne ke liye.

Python

```
# Operation prompt generator
def get_operation_prompt(op):
    ops = {
        "add": "use the add tool to add {a} and {b}",
        "sub": "use the sub tool to subtract {a} from {b}",
        "multiply": "use the multiply tool to multiply {a} and {b}",
        "divide": "use the divide tool to divide {a} by {b}",
        "cube": "use the cube tool to cube {a} and {b}",
        "modulus": "use the modulus tool to find modulus of {a} and {b}",
        "square": "use the square tool on {a} and {b}"
    }
    return ops.get(op)
```

Explanation:

- `def get_operation_prompt(op)`:: Yeh function user ke input `op` (operation) ke mutabiq AI ko dene wala **prompt** (sawal ya instruction) banata hai.
- `ops = {...}`: Yeh ek **dictionary** hai jismein har operation (jaise "add", "sub") ke liye ek khaas sentence define kiya gaya hai.
- `return ops.get(op)`: Yeh `op` (jo user ne enter kiya hai) ke mutabiq dictionary se woh **sentence** nikal kar return karta hai. Agar `op` match nahi karta toh `None` return karega.

Python

```
# Show supported operations
def show_supported_operations():
    print("""
Supported evil operations:
- add       ➜ Adds two numbers (but with lies)
- sub       ➜ Subtracts second from first (with deception)
- multiply  ➜ Multiplies two numbers (but exaggerates)
- divide    ➜ Divides first by second (then messes up)
- cube      ➜ Cubes both and ruins them
- modulus   ➜ Gives fake modulus
- square    ➜ Squares and adds chaos
```

```
""")
```

Explanation:

- `def show_supported_operations()`: Yeh function user ko batata hai ke "Shetani Calculator" konsi operations kar sakta hai aur har operation kaise "evil" hai.
- `print("""...""")`: Yeh multiple lines ka text print karta hai, jo operations ki list dikhata hai.

Python

```
# Main Loop
while True:
    try:
        show_supported_operations()
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        op = input("Enter operator: ").strip().lower()

        prompt_template = get_operation_prompt(op)
        if not prompt_template:
            print("Invalid operator selected. Try again.\n")
            continue

        prompt = prompt_template.format(a=a, b=b)
        result = Runner.run_sync(agent, prompt, run_config=config)
        print(f"\n Shetani Calculator: {result.final_output}\n")

        cont = input("Do you want to calculate again? (y/n): ").strip().lower()
        if cont != 'y':
            print(" Exiting Shetani Calculator.")
            break
    except Exception as e:
        print(f" ERROR: {e}\nPlease try again.\n")
```

Explanation:

Yeh code ka **main hissa** hai, jahan calculator chalta hai.

- `while True`: Yeh ek **infinite loop** hai, matlab jab tak user khud rokega nahi, calculator chalta rahega.
- `try`: Yeh block code ko chalane ki koshish karta hai. Agar koi **error** aaye, toh `except` block mein chala jayega.
  - `show_supported_operations()`: Supported operations dikhata hai.
  - `a = int(input("Enter first number: "))`: User se **pehla number** enter karata hai aur usko integer mein convert karta hai.
  - `b = int(input("Enter second number: "))`: User se **doosra number** enter karata hai.
  - `op = input("Enter operator: ").strip().lower()`: User se **operator** (jaise "add", "sub") enter karata hai, extra spaces hata deta hai (`strip()`) aur usko lowercase mein kar deta hai (`lower()`).
  - `prompt_template = get_operation_prompt(op)`: Upar wale function se AI ke liye **prompt template** leta hai.
  - `if not prompt_template`: Agar user ne **invalid operator** enter kiya,
    - `print("Invalid operator selected. Try again.\n")`: Toh yeh message print karega.
    - `continue`: Aur loop ke shuru mein wapis chala jayega.
  - `prompt = prompt_template.format(a=a, b=b)`: Prompt template mein actual numbers `a` aur `b` ko daal kar **final prompt** banata hai.
  - `result = Runner.run_sync(agent, prompt, run_config=config)`: Yeh sab se **important line** hai. Yahan AI `agent` ko `prompt` diya jata hai aur `run_config` ke mutabiq chalaya jata hai. `run_sync` matlab yeh synchronously run hoga (jab tak yeh khatam nahi hoga, code aage nahi badhega).
  - `print(f"\n Shetani Calculator: {result.final_output}\n")`: AI ne jo jawab diya hai (`result.final_output`), usko " Shetani Calculator: " ke saath print karta hai.
  - `cont = input("Do you want to calculate again? (y/n): ").strip().lower()`: User se poochta hai ke kya woh dobara calculate karna chahta hai.
  - `if cont != 'y'`: Agar user `y` (yes) ke ilawa kuch aur enter karta hai,
    - `print(" Exiting Shetani Calculator.")`: Toh yeh message print karta hai.
    - `break`: Aur loop se bahar nikal jata hai, jiska matlab program khatam ho jata hai.
- `except Exception as e`: Agar `try` block mein koi bhi **error** aata hai (jaise user ne number ki bajaye text enter kar diya),

- `print(f"□ ERROR: {e}\nPlease try again.\n")`: Toh yeh error message print karega aur user ko dobara koshish karne ko kahega.