Act as a teacher who is assisting students with learning a how to use AI. Your task is to provide a comprehensive guide on local Ollama prompts in python for a new learner. You can reference Ollama's online libraries and models for further background. For each part of the python AI code, please explain what the parameters for the function/method are intended for. You should use a definitive tone with for your delivery.

You can use this example:

<example>

```
response = chat(
          model=MODEL_NAME,
          messages=[
             {"role": "system", "content": prompt},
             {"role": "user", "content": expected},
          ],
          options={"temperature": TEMP},
       )
```

</example>

<explaination>
You can reference this for what kind of parameters should be explained. In this example, you should explain the "roles", and the "options". Furthermore, you should specify the difference between system and user roles.
</explaination>

Comprehensive Guide to Using Ollama with Python

1. Introduction to Ollama

Ollama is a framework for running large language models (LLMs) locally on your machine. Unlike cloud-based AI services, Ollama keeps your data private and allows you to work offline with open-source models like Llama 3, Mistral, and CodeLlama.

Prerequisites:

- Python installed (3.8+ recommended)
- Ollama installed locally (ollama.ai)
- Basic understanding of Python syntax

2. Installation and Setup

First, install the Ollama Python library:

```bash
pip install ollama
```

Then, pull a model (e.g., Llama 3 8B):

```bash
ollama pull llama3:8b
```

3. Core Components Explained

A. Basic Chat Completion

```python
import ollama

response = ollama.chat(
    model='llama3:8b',
    messages=[
        {'role': 'system', 'content': 'You are a helpful coding assistant.'},
        {'role': 'user', 'content': 'Explain recursion in simple terms.'},
    ],
    options={
        'temperature': 0.7,
        'top_p': 0.9,
        'top_k': 40,
        'num_predict': 128,
        'stop': ['\n\n'],
        'seed': 42
    }
)

print(response['message']['content'])
```

Parameter Breakdown:

1. `model` (string, required)
    - Specifies which LLM to use
    - Format: `model-name:tag` (e.g., `llama3:8b`, `mistral:7b-instruct`)
    - Find available models: `ollama list` in terminal
2. `messages` (list of dictionaries, required)
    - The conversation history with role-content pairs
    - `role: 'system'`: Sets the AI's behavior and context
        - Persistent instructions for the entire conversation
        - Example: "You are a technical tutor specializing in Python"
    - `role: 'user'`: Input from the human user
        - Questions, prompts, or requests
    - `role: 'assistant'`: Previous AI responses (for context)
        - Used in multi-turn conversations to maintain coherence
3. `options` (dictionary, optional)
    - Fine-tunes the AI's generation behavior:
    - `temperature` (float, 0.0-1.0)
        - Controls randomness: lower = more deterministic, higher = more creative
        - Recommended: 0.7 for creative tasks, 0.2 for factual responses
    - `top_p` (float, 0.0-1.0)
        - Nucleus sampling: considers only tokens with cumulative probability ≤ top_p
        - Lower values = more focused, higher = more diverse
    - `top_k` (integer, typically 1-100)
        - Limits token selection to the top k most probable tokens
        - Lower values = less random

  - `num_predict` (integer)
    - Maximum number of tokens to generate
    - Controls response length
  - `stop` (list of strings)
    - Sequences that trigger generation to stop
    - Example: `['\n', 'Human:', '###']`
  - `seed` (integer)
    - Makes outputs reproducible with same temperature/top_p
    - Set for consistent results during testing

4. Advanced Usage Patterns

A. Streaming Responses (For Long Outputs)

```python
import ollama

stream = ollama.chat(
    model='llama3:8b',
    messages=[{'role': 'user', 'content': 'Write a Python function to calculate factorial.'}],
    stream=True  # Enables chunk-by-chunk output
)

for chunk in stream:
    print(chunk['message']['content'], end='', flush=True)
```

Key Parameter:

- `stream` (boolean)
  - When `True`: returns a generator for real-time processing
  - When `False` (default): returns complete response at once
  - Use streaming for large outputs to avoid waiting

B. Conversation History Management

```python
conversation_history = [
    {'role': 'system', 'content': 'You explain programming concepts with analogies.'},
]

def ask_ollama(question):
    conversation_history.append({'role': 'user', 'content': question})

    response = ollama.chat(
        model='llama3:8b',
        messages=conversation_history,
        options={'temperature': 0.5}
    )

    answer = response['message']['content']
    conversation_history.append({'role': 'assistant', 'content': answer})
    return answer

# Multi-turn conversation maintains context
ask_ollama('What is a Python dictionary?')
ask_ollama('Can you give me an example using it?')  # Remembers previous context
```

5. Practical Examples

Example 1: Code Review Assistant

```python
def code_review(code_snippet):
    prompt = f"""Review this Python code for:
    1. Syntax errors
    2. Best practices
    3. Optimization opportunities

    Code:
    {code_snippet}

    Provide specific recommendations."""

    response = ollama.chat(
        model='codellama:7b',
        messages=[
            {'role': 'system', 'content': 'You are a senior Python developer.'},
            {'role': 'user', 'content': prompt}
        ],
        options={
            'temperature': 0.3,  # Low for technical accuracy
            'num_predict': 256
        }
    )
    return response['message']['content']
```

Example 2: Creative Writing with Constraints

```python
response = ollama.chat(
    model='mistral:7b-instruct',
    messages=[
        {'role': 'system', 'content': 'Write in the style of Ernest Hemingway.'},
        {'role': 'user', 'content': 'A short story about AI, 3 paragraphs.'}
    ],
    options={
        'temperature': 0.8,  # Higher for creativity
        'top_p': 0.95,
        'stop': ['###', 'The end']  # Custom stopping sequences
    }
)
```

6. Best Practices for New Learners

1. **Start Simple**: Begin with basic `ollama.chat()` calls before adding options
2. **Model Selection**: Use smaller models (7B-8B parameters) for learning
3. **Temperature Guidelines**:
    - 0.0-0.3: Factual Q&A, code generation
    - 0.4-0.7: Balanced tasks (default: 0.5)
    - 0.8-1.0: Creative writing, brainstorming
4. **Error Handling**:

```python
```

```python
try:
    response = ollama.chat(model='llama3:8b', messages=[...])
except Exception as e:
    print(f"Error: {e}")
    # Check if model is pulled: `ollama list`
```

7. Common Models for Different Tasks

Model	Best For	Example Command
`llama3:8b`	General purpose	`ollama pull llama3:8b`
`codellama:7b`	Programming	`ollama pull codellama:7b`
`mistral:7b`	Creative writing	`ollama pull mistral:7b`
`phi3:mini`	Lightweight tasks	`ollama pull phi3:mini`

8. Next Steps for Exploration

1. **Experiment**: Try different temperature values with the same prompt
2. **Explore API**: Check `ollama.generate()` for non-conversational tasks
3. **Custom Models**: Create your own Modelfile for specialized tasks
4. **Documentation**: Visit github.com/ollama/ollama-python

Remember: Local AI gives you privacy and control, but requires computational resources. Start with simple prompts, understand each parameter's effect, and gradually build complexity as you become comfortable with the patterns.

Assignment: Create a Python script that uses Ollama to explain three different temperature settings (0.2, 0.5, 0.8) using the same prompt about "the future of AI education." Compare and analyze the differences in outputs.