Act a life-long professional 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

Prerequisites and Setup

Before diving into the code, ensure you have:

1. **Ollama installed** and running locally (download from ollama.ai)
2. **Python package** installed: `pip install ollama`
3. **A model pulled** locally: `ollama pull llama3.2` (or your preferred model)

Core Components Explained

1. Basic Completion Request

```python
import ollama

response = ollama.generate(
    model="llama3.2",           # Parameter: Model name (must be pulled locally)
    prompt="Explain quantum computing",  # Parameter: Input text for completion
    stream=False,               # Parameter: Whether to stream response incrementally
    options={                   # Parameter: Generation configuration
        "temperature": 0.7,     # Controls randomness (0.0 = deterministic, 1.0 = cre
ative)
        "top_p": 0.9,          # Nucleus sampling parameter
        "top_k": 40,           # Limits token selection to top K options
        "num_predict": 128,    # Maximum tokens to generate
        "repeat_penalty": 1.1, # Penalty for repeating tokens
        "seed": 42             # Random seed for reproducibility
    }
)

print(response['response'])    # Access the generated text
```

Key Parameters Explained:

- `model`: Specifies which locally-stored Ollama model to use (e.g., "llama3.2", "mistral", "codellama")
- `prompt`: The text input that the model will complete or respond to
- `stream`: When `True`, yields responses chunk by chunk; useful for real-time display
- `options`: Dictionary containing fine-tuning parameters for generation quality and behavior

2. Chat Completion with Role Management

```python
```

```python
import ollama

response = ollama.chat(
    model="llama3.2",
    messages=[
        {
            "role": "system",      # Parameter: Sets context/behavior for entire conversation
            "content": "You are a helpful physics tutor. Explain concepts at a high school level."
        },
        {
            "role": "user",        # Parameter: Current user input/query
            "content": "What is Newton's third law?"
        },
        {
            "role": "assistant",   # Parameter: Previous assistant responses (for context)
            "content": "Newton's third law states: For every action, there is an equal and opposite reaction."
        },
        {
            "role": "user",
            "content": "Give me a real-world example of this."
        }
    ],
    format="json",                 # Parameter: Constrains output to JSON format
    stream=False,
    options={
        "temperature": 0.3,        # Lower temperature for more focused, educational responses
        "num_ctx": 4096,          # Context window size (tokens)
        "num_gpu": 1,             # GPU layers to use (if available)
        "stop": ["\n\n"]          # Sequence that signals the model to stop generating
    }
)

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

Role System Explained:

- **System**: Sets the AI's behavior, expertise, and constraints. This persists throughout the conversation and guides how the assistant responds. Use this for instructions, not conversation.
- **User**: Represents human inputs or queries. Each user message should be a self-contained question or statement.
- **Assistant**: Previous AI responses. Including these maintains conversation history and context.

3. Streaming Responses (Real-time Output)

```python
import ollama

stream = ollama.chat(
    model="llama3.2",
    messages=[{"role": "user", "content": "Write a short poem about AI"}],
    stream=True,                 # Enables streaming response
    options={"temperature": 0.8}
)

for chunk in stream:             # Iterate through response chunks
    content = chunk['message']['content']
    print(content, end='', flush=True)  # Print without newline, flush buffer
```

Streaming Benefits:

- Displays text as it's generated (better user experience)
- Lower latency for first token
- Can process long responses without waiting for completion

4. Advanced Configuration with Custom Models

```python
import ollama

# Create a custom model from an existing one
ollama.create(
    model="my-custom-llama",
    modelfile="""
    FROM llama3.2
    PARAMETER temperature 0.5
    PARAMETER top_p 0.95
    SYSTEM "You are a technical documentation expert."
    """
)

# Use the custom model
response = ollama.generate(
    model="my-custom-llama",
    prompt="Document this Python function: def calculate_sum(a, b): return a + b",
    options={
        "num_predict": 256,
        "repeat_penalty": 1.2,    # Higher penalty reduces repetition
        "mirostat": 2,            # Advanced sampling method (0=off, 1=Mirostat, 2=Mi
rostat 2.0)
        "mirostat_tau": 5.0,      # Target perplexity for Mirostat
        "mirostat_eta": 0.1       # Learning rate for Mirostat
    }
)
```

5. Embedding Generation for Semantic Analysis

```python
import ollama

# Generate embeddings for text
embeddings = ollama.embeddings(
    model="llama3.2",             # Some models support embeddings
    prompt="The quick brown fox jumps over the lazy dog"
)

vector = embeddings['embedding']  # Returns a list of floats (vector representation)
print(f"Embedding dimension: {len(vector)}")
```

Embedding Applications:

- Semantic search

- Text classification
- Clustering similar documents
- Recommendation systems

Best Practices and Configuration Guide

Temperature Settings:

- **0.0-0.3**: Factual, deterministic responses (Q&A, documentation)
- **0.4-0.7**: Balanced creativity (general chat, brainstorming)
- **0.8-1.0**: Highly creative (storytelling, poetry)

Context Management:

```python
# Manage long conversations
response = ollama.chat(
    model="llama3.2",
    messages=messages[-10:],  # Keep last 10 messages to avoid context overflow
    options={"num_ctx": 8192}  # Match model's maximum context window
)
```

Error Handling:

```python
import ollama
from ollama import ResponseError

try:
    response = ollama.chat(
        model="llama3.2",
        messages=[{"role": "user", "content": "Your query here"}],
        options={"temperature": 0.5}
    )
except ResponseError as e:
    print(f"Error: {e.error}")
    # Common errors: model not found, context overflow, server not running
except Exception as e:
    print(f"Unexpected error: {e}")
```

Practical Implementation Pattern

```python
class OllamaAssistant:
    def __init__(self, model_name="llama3.2", system_prompt=None):
        self.model = model_name
        self.messages = []
        if system_prompt:
            self.messages.append({"role": "system", "content": system_prompt})

    def chat(self, user_input, temperature=0.7, max_tokens=150):
        """Complete chat cycle with history maintenance"""
        self.messages.append({"role": "user", "content": user_input})

        response = ollama.chat(
            model=self.model,
            messages=self.messages,
            options={
                "temperature": temperature,
                "num_predict": max_tokens,
                "repeat_penalty": 1.1
            }
        )

        assistant_response = response['message']['content']
        self.messages.append({"role": "assistant", "content": assistant_response})

        # Optional: Trim conversation history to prevent context overflow
        if len(self.messages) > 20:
            self.messages = [self.messages[0]] + self.messages[-19:]

        return assistant_response

# Usage
assistant = OllamaAssistant(
    system_prompt="You are a Python programming assistant. Provide code examples with explanations."
)
response = assistant.chat("How do I read a CSV file in Python?")
```

Further Resources

1. **Official Documentation:** github.com/ollama/ollama-python
2. **Available Models:** Run `ollama list` locally or check ollama.ai/library
3. **Model Customization:** See Modelfile reference for creating specialized variants
4. **API Alternatives:** Ollama also provides REST API for language-agnostic integration

Remember: Local AI with Ollama offers privacy, cost control, and customization. Start with simpler models for learning, experiment with parameters systematically, and always validate critical outputs. The key to mastery is iterative experimentation