Python Jumpstart: Coding Fundamentals for the AI Era

Michael Borck

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1. Your Python Journey Map: Guide to Chapters and Resources

2. Just Enough Python

This series introduces the essential components of the programming language Python. The focus is on the programming language and not on the craft of programming. The content does not discuss everything about Python but provides a foundation so you can take it further. The material has been prepared for students in the Faculty of Business and Law at Curtin University but has been updated to include modern AI-assisted programming techniques.

2.1. Core Python Fundamentals

1. Python in the Age of AI
2. Basic Python Syntax
3. Values
4. Variables
5. Output
6. Input
7. Operators

2.2. Functions and Control Flow

8. Using Functions
9. Creating Functions
10. Making Decisions

2.3. Data Structures and Iteration

2.4. Working with Data and Files

2.5. Code Quality and Organization

2.6. Practical Python Usage

2.7. Python in the AI Era

24. AI Programming Assistants
25. Python for AI Integration
26. AI Assistance Tips
27. Intentional Prompting for Python Developers

2.8. Project: Build Your Own AI Chatbot

- Building Your AI-Enhanced Python Chatbot

2.9. Similar Python Tutorials

Here are some free beginner tutorials that inspired this series. They have a data science focus but you may find them useful in your learning journey.

- Introduction to Python (Microsoft)
- Python (Kaggle)
- Whirlwind Tour of Python
- Learn Python in 1 hour (Video)
- Learn Python Basics for Data Analysis

Part I.

Core Python Fundamentals

3. Python in the Age of AI: Coding with Digital Collaborators

4. Syntax Safari: Navigating the Python Language Landscape

5. Value Voyage: Exploring Python's Data Universe

6. Variable Vault: Naming and Storing Your Digital Treasures

7. Output Odyssey: Making Your Code Speak to the World

8. Input Inception: Getting Data from Users into Your Programs

9. Operator's Manual: Mastering Python's Mathematical and Logical Tools

Part II.

Functions and Control Flow

10. Function Fiesta: Using Python's Pre-built Code Blocks

11. Function Factory: Crafting Your Own Reusable Code Magic

12. Decision Director: Guiding Your Program's Path with If Statements

Part III.

Data Structures and Iteration

13. List Laboratory: Organizing Data in Python's Most Versatile Container

14. Going Loopy: Repeating Code Without Losing Your Mind

15. String Theory: Manipulating Text in the Python Universe

16. Dictionary Detectives: Mastering Python's Key-Value Pairs

Part IV.

Working with Data and Files

17. File Frontier: Reading and Writing Data to Permanent Storage

Part V.

Code Quality and Organization

18. Error Embassy: Understanding and Handling Exceptions with Grace

19. Debugging Detectives: Finding and Fixing Code Mysteries

20. Test Kitchen: Ensuring Your Code Works as Intended

21. Module Mastery: Organizing Your Code for Growth and Reuse

22. Orientating Your Objects: Building Digital Models of Real-World Things

Part VI.

Practical Python Usage

23. Python Pilot: How to Execute Your Code in Different Environments

24. Installation Station: Setting Up Python and Required Libraries

25. Help Headquarters: Finding Answers When You Get Stuck

Part VII.

Python in the AI Era

26. AI Programming Assistants: Coding with Digital Colleagues

27. Python AI Integration: Connecting Your Code to Intelligent Services

28. AI Assistance Tips: Maximizing Your Machine Learning Mentors

29. Intentional Prompting: Speaking the Language of AI Assistants

Part VIII.

Project: Build Your Own AI Chatbot

30. Chatbot Construction Site: Building Your AI-Enhanced Python Conversation Partner

31. Building Your AI-Enhanced Python Chatbot

This guide outlines an incremental project that spans multiple chapters in the book. As you progress through the Python concepts, you'll apply your knowledge to build a chatbot that becomes increasingly sophisticated.

31.1. Project Overview

The project follows this progression:

1. **Basic Rule-Based Chatbot** (Chapters 1-7)
    - Simple input/output with hardcoded responses
    - Basic string manipulation
    - Introduction to variables and operators
2. **Structured Chatbot** (Chapters 8-14)
    - Using functions to organize code
    - Implementing decision logic with conditionals
    - Storing conversation history in lists
    - Managing response templates with dictionaries
3. **Persistent Chatbot** (Chapters 15-20)
    - Saving and loading chat history from files
    - Error handling for robust user interaction

- Modular design with functions in separate modules
- Object-oriented approach for a more maintainable chatbot

4. **AI-Enhanced Chatbot** (Chapters 21-26)
   - Integration with AI services for smarter responses
   - Using modern Python libraries and tools
   - Advanced conversation understanding

31.2. Chapter-by-Chapter Implementation

This guide provides code snippets to implement at each stage of your learning journey. Add these to your chatbot as you progress through the related chapters.

31.2.1. Stage 1: Basic Rule-Based Chatbot

After Chapter 4: Variables

```
# Simple chatbot using variables
bot_name = "PyBot"
user_name = input("Hello! I'm " + bot_name + ". What's your name? ")
print("Nice to meet you, " + user_name + "!")
```

After Chapter 5: Output

```
# Enhanced output formatting
print(f"Hello {user_name}! I'm {bot_name}, a simple chatbot.")
print(f"I was created as a learning project in Python.")
print(f"I don't know much yet, but I'll get smarter as you learn more Pyth
```

After Chapter 7: Operators

```python
# Using operators for basic logic
user_input = input("Ask me a question: ")
response = "I'm not sure how to answer that yet."

if "hello" in user_input.lower():
    response = f"Hello there, {user_name}!"
elif "name" in user_input.lower():
    response = f"My name is {bot_name}!"
elif "age" in user_input.lower():
    response = "I was just created, so I'm very young!"

print(response)
```

31.2.2. Stage 2: Structured Chatbot

After Chapter 9: Creating Functions

```python
def get_response(user_input):
    """Return a response based on the user input."""
    user_input = user_input.lower()

    if "hello" in user_input:
        return f"Hello there, {user_name}!"
    elif "how are you" in user_input:
        return "I'm just a computer program, but thanks for asking!"
    elif "name" in user_input:
        return f"My name is {bot_name}!"
    elif "bye" in user_input or "goodbye" in user_input:
        return "Goodbye! Have a great day!"
    else:
        return "I'm not sure how to respond to that yet."
```

```
# Main chat loop
print(f"Hello! I'm {bot_name}. Type 'bye' to exit.")
user_name = input("What's your name? ")
print(f"Nice to meet you, {user_name}!")

while True:
    user_input = input(f"{user_name}> ")
    if user_input.lower() == "bye":
        print(f"{bot_name}> Goodbye, {user_name}!")
        break

    response = get_response(user_input)
    print(f"{bot_name}> {response}")
```

After Chapter 11: Lists

```
# Add this to your chatbot code to track conversation history
conversation_history = []

def save_to_history(speaker, text):
    """Save an utterance to conversation history."""
    conversation_history.append(f"{speaker}: {text}")

def show_history():
    """Display the conversation history."""
    print("\n----- Conversation History -----")
    for entry in conversation_history:
        print(entry)
    print("-------------------------------\n")

# Then in your main loop, update to use these functions:
while True:
    user_input = input(f"{user_name}> ")
    save_to_history(user_name, user_input)

    if user_input.lower() == "bye":
        response = f"Goodbye, {user_name}!"
        print(f"{bot_name}> {response}")
        save_to_history(bot_name, response)
        break
    elif user_input.lower() == "history":
        show_history()
        continue

    response = get_response(user_input)
    print(f"{bot_name}> {response}")
    save_to_history(bot_name, response)
```

After Chapter 14: Dictionaries

```
# Using dictionaries for smarter response patterns
response_patterns = {
    "greetings": ["hello", "hi", "hey", "howdy", "hola"],
    "farewells": ["bye", "goodbye", "see you", "cya", "farewell"],
    "gratitude": ["thanks", "thank you", "appreciate"],
    "bot_questions": ["who are you", "what are you", "your name"],
    "user_questions": ["how are you", "what's up", "how do you feel"]
}

response_templates = {
    "greetings": [f"Hello, {user_name}!", f"Hi there, {user_name}!", "Great to see you ag
    "farewells": ["Goodbye!", "See you later!", "Until next time!"],
    "gratitude": ["You're welcome!", "Happy to help!", "No problem at all."],
    "bot_questions": [f"I'm {bot_name}, your chatbot assistant!", "I'm just a simple Pyth
    "user_questions": ["I'm just a program, but I'm working well!", "I'm h
}

import random

def get_response(user_input):
    """Get a more sophisticated response using dictionaries."""
    user_input = user_input.lower()

    # Check each category of responses
    for category, patterns in response_patterns.items():
        for pattern in patterns:
            if pattern in user_input:
                # Return a random response from the appropriate category
                return random.choice(response_templates[category])

    # Default response if no patterns match
    return "I'm still learning. Can you tell me more?"
```

31.2.3. Stage 3: Persistent Chatbot

After Chapter 15: Files

```
# Add to your chatbot the ability to save and load conversation history
import datetime

def save_conversation():
    """Save the current conversation to a file."""
    timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"chat_with_{user_name}_{timestamp}.txt"
    with open(filename, "w") as f:
        f.write(f"Conversation with {bot_name} and {user_name}\n")
        f.write(f"Date: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n\n")

        for entry in conversation_history:
            f.write(f"{entry}\n")

    return filename

# Add to your main loop:
while True:
    # ... existing code ...

    if user_input.lower() == "save":
        filename = save_conversation()
        print(f"{bot_name}> Conversation saved to {filename}")
        continue
```

After Chapter 16: Errors and Exceptions

```
# Add error handling to your chatbot
def load_conversation(filename):
    """Load a previous conversation from a file."""
    try:
        with open(filename, "r") as f:
            lines = f.readlines()

        print("\n----- Loaded Conversation -----")
        for line in lines:
            print(line.strip())
        print("-------------------------------\n")
        return True
    except FileNotFoundError:
        print(f"{bot_name}> Sorry, I couldn't find that file.")
        return False
    except Exception as e:
        print(f"{bot_name}> An error occurred: {str(e)}")
        return False

# Add to your main loop:
while True:
    # ... existing code ...

    if user_input.lower().startswith("load "):
        filename = user_input[5:].strip()
        load_conversation(filename)
        continue
```

After Chapter 19: Modules and Packages

```
# Organize your chatbot into a module structure
# You would create these files:

# chatbot/response_manager.py
"""Functions for generating chatbot responses."""
import random

class ResponseManager:
    def __init__(self, bot_name):
        self.bot_name = bot_name
        self.response_patterns = {
            # ... your patterns here ...
        }

        self.response_templates = {
            # ... your templates here ...
        }

    def get_response(self, user_input, user_name):
        """Generate a response to the user input."""
        # Your response logic here

# chatbot/history_manager.py
"""Functions for managing conversation history."""
import datetime

class HistoryManager:
    def __init__(self):
        self.conversation_history = []

    def add_to_history(self, speaker, text):
        """Add a message to history."""
        self.conversation_history.append(f"{speaker}: {text}")

    def show_history(self):
        """Display the conversation history."""
        # Your display code here

    def save_conversation(self, user_name, bot_name):
        """Save the conversation to a file."""
        # Your save code here

# chatbot/main.py
"""Main chatbot interface."""
from chatbot.response_manager import ResponseManager
from chatbot.history_manager import HistoryManager
```

```python
def run_chatbot():
    """Run the main chatbot loop."""
    bot_name = "PyBot"
    response_manager = ResponseManager(bot_name)
    history_manager = HistoryManager()

    print(f"Hello! I'm {bot_name}. Type 'bye' to exit.")
    user_name = input("What's your name? ")
    print(f"Nice to meet you, {user_name}!")

    # Main chat loop
    while True:
        # Your chatbot logic here
```

After Chapter 20: Object-Oriented Python

```python
# Convert your chatbot to a fully object-oriented design

class Chatbot:
    """A simple chatbot that becomes smarter as you learn Python."""

    def __init__(self, name="PyBot"):
        self.name = name
        self.user_name = None
        self.conversation_history = []
        self.response_patterns = {
            # ... your patterns ...
        }
        self.response_templates = {
            # ... your templates ...
        }
    def greet(self):
        """Greet the user and get their name."""
        print(f"Hello! I'm {self.name}. Type 'bye' to exit.")
        self.user_name = input("What's your name? ")
        print(f"Nice to meet you, {self.user_name}!")

    def get_response(self, user_input):
        """Generate a response to the user input."""
        # Your response logic here

    def add_to_history(self, speaker, text):
        """Add a message to the conversation history."""
        # Your history code here

    def save_conversation(self):
        """Save the conversation to a file."""
        # Your save code here

    def load_conversation(self, filename):
        """Load a conversation from a file."""
        # Your load code here

    def run(self):
        """Run the main chatbot loop."""
        self.greet()

        while True:
            # Your main loop logic here

# To use:
if __name__ == "__main__":
    bot = Chatbot()
    bot.run()
```

31.2.4. Stage 4: AI-Enhanced Chatbot

After Chapter 25: Python for AI Integration

```
# Enhance your chatbot with AI capabilities
import os
from dotenv import load_dotenv
import openai  # You'll need to pip install openai

# Load API key from environment variable
load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")

class AIEnhancedChatbot(Chatbot):
    """A chatbot enhanced with AI capabilities."""

    def __init__(self, name="AI-PyBot"):
        super().__init__(name)
        self.ai_mode = False
        self.conversation_context = []

    def toggle_ai_mode(self):
        """Toggle between rule-based and AI-powered responses."""
        self.ai_mode = not self.ai_mode
        return f"AI mode is now {'on' if self.ai_mode else 'off'}"

    def get_ai_response(self, user_input):
        """Get a response from the OpenAI API."""
        # Add to conversation context
        self.conversation_context.append({"role": "user", "content": user_input})

        try:
            # Get response from OpenAI
            response = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=[
                    {"role": "system", "content": f"You are {self.name}, a helpful assist
                    *self.conversation_context
                ]
            )

            # Extract and save the assistant's response
            ai_response = response.choices[0].message["content"]
            self.conversation_context.append({"role": "assistant", "content": ai_response

            # Keep context window manageable (retain last 10 exchanges)
            if len(self.conversation_context) > 20:
                self.conversation_context = self.conversation_context[-20:]

            return ai_response

        except Exception as e:
            return f"AI error: {str(e)}"

    def get_response(self, user_input):
        """Get a response using either rule-based or AI approach."""
        if user_input.lower() == "ai mode":
            return self.toggle_ai_mode()

        if self.ai_mode:
            return self.get_ai_response(user_input)
        else:
            return super().get_response(user_input)
```

31.3. Project Challenges and Extensions

As you become more comfortable with Python, try these challenges to enhance your chatbot further:

1. **Sentiment Analysis**: Analyze the sentiment of user messages and adjust responses accordingly.
2. **Web Integration**: Make your chatbot accessible via a simple web interface using Flask.
3. **Voice Capabilities**: Add text-to-speech and speech-to-text capabilities.
4. **Knowledge Base**: Create a system for your chatbot to learn facts and retrieve them when asked.
5. **Multi-language Support**: Add the ability to detect and respond in different languages.

31.4. How to Use This Guide

1. Work through the book chapters in order
2. When you reach a chapter mentioned in this guide, implement the corresponding chatbot enhancements
3. Test and experiment with the chatbot after each implementation
4. By the end of the book, you'll have a sophisticated AI-enhanced chatbot built entirely by you!

Remember: This project is meant to be flexible. Feel free to customize your chatbot, add your own features, and make it truly yours!