ITEC5025 Week 5

Data Structures: Lists and Dictionaries

Complete Repository Guide

A Simple English Explanation of the

Hypotify Clinical Insights Chatbot

Student: Shruti Malik

Course: ITEC5025 - Information Technology Fundamentals

Date: February 8, 2026

Table of Contents

1. Introduction to the Project

This repository contains a complete Python chatbot project created for Week 5 of the ITEC5025 course. The main goal of this project is to show how two important programming concepts work: **Lists** and **Dictionaries**.

Think of this project like a digital filing system for a hospital. Just like a real hospital keeps records of patients in filing cabinets, this program keeps patient information organized in the computer's memory using lists and dictionaries.

What Makes This Project Special?

• It can store information about multiple patients

• It can search for specific patients quickly

• It can translate medical terms into different languages

• It can analyze patient data and show statistics

• It can understand what you're asking for in natural language

The chatbot is named **Hypotify**, and it acts like a smart assistant that helps doctors and nurses manage patient information efficiently.

2. What is a Chatbot?

A chatbot is a computer program that can have conversations with people. Instead of clicking buttons or filling out forms, you can simply type questions or commands, and the chatbot responds just like you're texting with a friend.

Simple Chatbot Example:

You: What's the weather like?
Chatbot: It's sunny and 72 degrees today!

How Does Our Medical Chatbot Work?

Our chatbot understands medical-related questions and commands. For example:

You: How many patients do we have?

Bot: We have 5 patients in the database.

You: Show me Spanish-speaking patients

Bot: Found 1 patient who speaks Spanish.

You: What's the average age?

Bot: The average patient age is 34.6 years.

The chatbot is programmed to recognize keywords in your questions and provide the right information from the patient database.

3. Understanding Data Structures

Data structures are ways to organize and store information in a computer program. Think of them like different types of containers for storing things.

Real-World Analogy:

Imagine you're organizing your room:

• **A List** is like a bookshelf where books are arranged in order. You can add books, remove books, or find a specific book by looking through them one by one.

• **A Dictionary** is like a phone book where each name has a phone number. Instead of looking through every entry, you can jump directly to the name you want and find the number.

Why Are Data Structures Important?

• **Speed:** They help computers find information quickly

• **Organization:** They keep data neat and easy to manage

• **Efficiency:** They use computer memory wisely

• **Flexibility:** They make it easy to add or remove information

In this project, we use lists and dictionaries to store patient information so the chatbot can quickly answer questions about patients.

4. Lists Explained Simply

A **list** in Python is an ordered collection of items. Think of it as a shopping list or a to-do list where items are numbered in order.

Simple Example:

Imagine a grocery list:
1. Milk
2. Eggs
3. Bread
4. Butter

In Python, this would look like:

```
grocery_list = ['Milk', 'Eggs', 'Bread', 'Butter']
```

What Can You Do With Lists?

• **Add items:** grocery_list.append('Cheese') adds Cheese to the end

Result: Now the list has 5 items

• **Remove items:** grocery_list.remove('Eggs') takes Eggs off the list

Result: Now the list has 4 items again

• **Count items:** len(grocery_list) tells you how many items

Result: This returns 4

• **Check if item exists:** 'Milk' in grocery_list

Result: This returns True because Milk is on the list

• **Access specific item:** grocery_list[0]

Result: This gets the first item: 'Milk' (counting starts at 0!)

Lists in Our Chatbot:

In the patient database, we use a list called **PATIENTS** to store all patient records. Each patient is an item in this list, making it easy to count patients, add new patients, or go through all patients one by one.

5. Dictionaries Explained Simply

A **dictionary** in Python is a collection of key-value pairs. Think of it like a real dictionary where you look up a word (the key) to find its definition (the value).

Simple Example:

Imagine you want to store phone numbers for your friends:

```
phone_book = {
  'Alice': '555-1234',
  'Bob': '555-5678',
  'Charlie': '555-9012'
}
```

In this example, the names (Alice, Bob, Charlie) are **keys**, and the phone numbers are **values**.

What Can You Do With Dictionaries?

• **Look up a value:** phone_book['Alice']

Result: Returns '555-1234' instantly

• **Add a new entry:** phone_book['David'] = '555-3456'

Result: Adds David with his phone number

• **Update a value:** phone_book['Alice'] = '555-9999'

Result: Changes Alice's number

• **Remove an entry:** del phone_book['Bob']

Result: Removes Bob from the phone book

• **Check if key exists:** 'Alice' in phone_book

Result: Returns True if Alice is in the book

Dictionaries in Our Chatbot:

Each patient in our system is stored as a dictionary. For example:

```
patient = {
  'patient_id': 'P001',
  'name': 'John Smith',
  'age': 45,
  'language': 'English',
  'gender': 'Male'
}
```

This makes it easy to access any piece of information about a patient. Want to know John's age? Just use patient['age'] and get 45 immediately!

6. The Patient Database System

The heart of our chatbot is the patient database. This is where all patient information is stored and organized using lists and dictionaries working together.

How It's Organized:

The database uses two main data structures:

1. PATIENTS List: A list that holds all patient records in order

```
PATIENTS = [patient1, patient2, patient3, patient4, patient5]
```

2. PATIENT_INDEX Dictionary: A dictionary for quick lookups by patient ID

```
PATIENT_INDEX = {
  'P001': patient1,
  'P002': patient2,
  'P003': patient3,
  'P004': patient4,
  'P005': patient5
}
```

Sample Patient Record:

Here's what a complete patient record looks like:

```
patient1 = {
  'patient_id': 'P001',
  'name': 'Emma Johnson',
  'age': 29,
  'language': 'English',
  'gender': 'Female',
  'race': 'Caucasian',
  'poverty_level': False,
  'medical_history': ['Asthma', 'Seasonal Allergies']
}
```

Why Use Both Lists and Dictionaries?

• **Lists are good for:** Counting all patients, showing all patients, adding patients in order

• **Dictionaries are good for:** Finding a specific patient quickly by their ID without searching through everyone

Using both together gives us the best of both worlds - we can work with all patients easily AND find specific patients instantly!

7. How the Chatbot Works

The chatbot follows a simple process when you interact with it. Think of it like having a conversation with a smart assistant.

The Conversation Flow:

Step 1 - You Type a Message:

You type something like "show me all patients"

Step 2 - Text Processing:

The chatbot reads your message and converts it to lowercase to understand it better

Step 3 - Keyword Detection:

It looks for important words like 'patients', 'count', 'show', 'translate', etc.

Step 4 - Action Selection:

Based on the keywords, it decides what action to take

Step 5 - Execute Action:

It performs the requested action (search database, calculate statistics, etc.)

Step 6 - Response:

It sends back a helpful response with the information you requested

Example Interaction:

You type: 'How many patients speak Spanish?'

1. Converts to lowercase: 'how many patients speak spanish?'

2. Finds keywords: 'patients', 'spanish'

3. Recognizes this is a language filter request

4. Searches through PATIENTS list for language == 'Spanish'

5. Counts the matches

6. Responds: 'Found 1 patient who speaks Spanish'

All of this happens in milliseconds! The chatbot can understand many different ways of asking the same question.

8. Main Features Overview

The Hypotify chatbot has several powerful features that work together to manage patient information. Let's look at each one:

1. Patient Database Management

This is the core feature that lets you work with patient records:

• View individual patient details

• List all patients at once

• Search for patients by language

• Add new patient records

• Update existing patient information

• Delete patient records

• Count total patients

2. Clinical Insights

This feature analyzes patient data to show helpful statistics:

• Language distribution (how many speak each language)

• Age statistics (youngest, oldest, average age)

• Gender demographics

• Race distribution

• Poverty level analysis

3. Translation Service

The chatbot can translate medical terms and phrases into over 100 languages using Google Translate. This helps communicate with patients who speak different languages.

4. Natural Language Processing

Using advanced AI libraries (NLTK and TensorFlow), the chatbot can understand what you mean even if you don't use exact commands. It's smart enough to figure out your intent!

9. Patient Database Management

Let's dive deeper into how the patient database management works. This is where lists and dictionaries really shine!

Viewing Patient Details

When you want to see information about a specific patient, you use their patient ID:

Command: 'show patient P001'
What happens: The chatbot looks in the PATIENT_INDEX dictionary using 'P001' as the key and instantly retrieves all information about that patient.

Listing All Patients

To see all patients:

Command: 'list all patients'
What happens: The chatbot uses a **for loop** to go through each patient in the PATIENTS list and displays their basic information.

The code behind this looks like:

```
for patient in PATIENTS:
  print(f"ID: {patient['patient_id']}")
  print(f"Name: {patient['name']}")
  print(f"Age: {patient['age']}")
```

Filtering by Language

You can find all patients who speak a specific language:

Command: 'find patients who speak Spanish'
What happens: The chatbot uses **list comprehension** (a special Python feature) to filter the PATIENTS list.

```
spanish_patients = [p for p in PATIENTS if p['language'] == 'Spanish']
```

This single line of code looks at every patient, checks if their language is Spanish, and creates a new list with only those patients. Pretty cool!

10. Clinical Insights Feature

The clinical insights feature analyzes all patient data to provide useful statistics. This helps healthcare providers understand their patient population better.

Language Distribution Analysis

This shows how many patients speak each language:

Example output:
Language Distribution (5 total patients):
  English: 3 patients (60.0%)
  Spanish: 1 patient (20.0%)
  Mandarin: 1 patient (20.0%)

How it works: The program creates a dictionary to count languages, then goes through all patients and adds to the count for each language. Finally, it calculates percentages.

Age Statistics

This provides insights about patient ages:

Example output:
Age Statistics:
  Average age: 34.6 years
  Youngest patient: 19 years
  Oldest patient: 52 years

How it works: The program collects all ages into a list, then uses Python's built-in functions: sum() for total, len() for count, min() for youngest, and max() for oldest.

Gender and Race Demographics

Similar to language distribution, the chatbot can show:

• How many patients are male, female, or other genders

• The racial breakdown of patients

• Percentage calculations for each category

Why This Matters:

These statistics help healthcare providers:

• Plan for language translation services

• Understand the age range of their patients

- Ensure care is appropriate for their demographic
- Identify underserved populations

11. Translation Feature

One of the most powerful features of the chatbot is its ability to translate medical information into different languages. This uses the Google Translate API.

How Translation Works:

When you ask for a translation:

1. You provide the text and target language

2. The chatbot sends this to Google's translation service

3. Google translates the text using artificial intelligence

4. The translated text comes back to the chatbot

5. The chatbot displays the translation to you

Example Usage:

English to Spanish:

Input: 'Please take this medication twice daily'

Output: 'Por favor tome este medicamento dos veces al día'

English to Mandarin:

Input: 'Your blood pressure is normal'

Output: '■■■■■■'

English to French:

Input: 'You need to schedule a follow-up appointment'

Output: 'Vous devez prendre un rendez-vous de suivi'

Supported Languages:

The chatbot supports translation to over 100 languages including:

• Spanish, French, German, Italian, Portuguese

• Mandarin Chinese, Japanese, Korean, Hindi, Arabic

• Russian, Polish, Dutch, Swedish, Turkish

• And many more!

Real-World Application:

This feature is incredibly useful in healthcare settings where patients speak different languages. Doctors can quickly translate medical instructions, making sure patients understand their treatment even if they don't share a common language.

12. Natural Language Processing

Natural Language Processing (NLP) is a technology that helps computers understand human language. Think of it as teaching the computer to read and understand like a person.

What is NLP?

Normally, computers only understand precise commands. But with NLP, they can understand natural, conversational language. For example:

Without NLP: You must type exact commands like 'SHOW_PATIENT_COUNT'
With NLP: You can type 'how many patients are there?' or 'what's the patient count?' or 'count patients' - all meaning the same thing!

How Our Chatbot Uses NLP:

The chatbot uses a library called **NLTK** (Natural Language Toolkit) to process your messages:

Tokenization:

Breaks your sentence into individual words. 'Show all patients' becomes ['Show', 'all', 'patients']

Lowercasing:

Converts everything to lowercase so 'Patients' and 'patients' are treated the same

Keyword Matching:

Looks for important words that indicate what you want to do

Intent Recognition:

Figures out what action you're requesting based on the keywords

TensorFlow Integration:

The chatbot is also set up to use **TensorFlow**, which is a powerful machine learning library. While the basic chatbot uses simple keyword matching, TensorFlow could be used to make it even smarter by:

• Learning from previous conversations

• Understanding context better

• Predicting what you might ask next

• Handling more complex questions

This shows the chatbot is built with future expansion in mind - it can grow smarter over time!

13. File Structure Explained

The project is organized into multiple Python files, each with a specific purpose. This is called **modular programming** - breaking a big program into smaller, manageable pieces.

Main Files:

main.py

The main program file. This is what you run to start the chatbot. It brings together all the other modules and handles the conversation flow.

patient_database.py

Contains the PATIENTS list and PATIENT_INDEX dictionary. This is where all patient data is stored and all database operations are defined.

clinical_insights.py

Handles all the statistical analysis - calculating averages, counting by category, and generating reports.

advanced_translation.py

Connects to Google Translate API and handles all translation requests.

nlp_processor.py

Contains the Natural Language Processing code using NLTK and TensorFlow.

medical_terms.py

A dictionary of medical terms and their definitions, useful for quick reference.

utils.py

Utility functions - helper functions used throughout the program.

test_all.py

Test code to make sure everything works correctly before using it with real data.

Why Split Into Multiple Files?

- **Organization:** Each file has a clear purpose, making it easy to find code
- **Reusability:** Functions can be used in multiple places without copying code
- **Maintenance:** Easier to fix bugs when code is organized
- **Collaboration:** Multiple people can work on different files simultaneously
- **Testing:** Each module can be tested independently

14. Code Examples - Lists

Let's look at actual code examples showing how lists are used in the chatbot.

Example 1: Creating the Patient List

```
# Start with an empty list
PATIENTS = []

# Add patient records one by one
PATIENTS.append(patient1)
PATIENTS.append(patient2)
PATIENTS.append(patient3)

# Now PATIENTS contains 3 patient records
```

Example 2: Counting Patients

```
# Get the total number of patients
total_patients = len(PATIENTS)
print(f'We have {total_patients} patients')

# Output: We have 5 patients
```

Example 3: Looping Through All Patients

```
# Go through each patient and print their name
for patient in PATIENTS:
  name = patient['name']
  print(f'Patient: {name}')

# Output:
# Patient: Emma Johnson
# Patient: Michael Chen
# Patient: Sofia Rodriguez
# Patient: James Wilson
# Patient: Aisha Patel
```

Example 4: Filtering with List Comprehension

```
# Find all patients over 40 years old
older_patients = [p for p in PATIENTS if p['age'] > 40]

# Find all female patients
female_patients = [p for p in PATIENTS if p['gender'] == 'Female']

# Find patients who speak English
english_speakers = [p for p in PATIENTS if p['language'] == 'English']
```

List comprehension is a powerful Python feature that makes filtering very concise. The syntax is: [item for item in list if condition]

15. Code Examples - Dictionaries

Now let's see how dictionaries are used to store and access patient information.

Example 1: Creating a Patient Dictionary

```
# Create a complete patient record
patient = {
  'patient_id': 'P001',
  'name': 'Emma Johnson',
  'age': 29,
  'language': 'English',
  'gender': 'Female',
  'race': 'Caucasian',
  'poverty_level': False
}
```

Example 2: Accessing Dictionary Values

```
# Get specific information
name = patient['name'] # Returns 'Emma Johnson'
age = patient['age'] # Returns 29
language = patient['language'] # Returns 'English'

# Print formatted information
print(f'{name} is {age} years old and speaks {language}')
# Output: Emma Johnson is 29 years old and speaks English
```

Example 3: The Patient Index Dictionary

```
# Create an index for fast lookups
PATIENT_INDEX = {}

# Add each patient to the index using their ID as the key
for patient in PATIENTS:
  patient_id = patient['patient_id']
  PATIENT_INDEX[patient_id] = patient

# Now we can find any patient instantly by ID
patient = PATIENT_INDEX['P001'] # Gets Emma Johnson immediately
```

Example 4: Updating Patient Information

```
# Update a patient's age
patient['age'] = 30

# Add a new field to the patient record
patient['phone'] = '555-1234'

# Update language preference
patient['language'] = 'Spanish'
```

Example 5: Checking if Key Exists

```
# Check if a patient ID exists
if 'P001' in PATIENT_INDEX:
  print('Patient found!')
else:
  print('Patient not found')

# Check if a patient has medical history
if 'medical_history' in patient:
  print(patient['medical_history'])
```

16. How to Use the Chatbot

Here's a step-by-step guide on how to use the Hypotify chatbot.

Getting Started:

Step 1: Install Python

Make sure you have Python 3.7 or newer installed on your computer

Step 2: Install Required Libraries

Open a terminal and run: pip install googletrans==4.0.0-rc1 nltk tensorflow numpy

Step 3: Download the Code

Get all the files from the GitHub repository

Step 4: Navigate to the Folder

Use terminal to go to the ITEC5025-WEEK5-Chatbot-Data-Structures folder

Step 5: Run the Chatbot

Type: python main.py

Common Commands:

'patients count' or 'how many patients'

Shows total number of patients

'show patient P001'

Displays details for patient P001

'list all patients'

Shows basic info for all patients

'patients language Spanish'

Finds patients who speak Spanish

'insights language'

Shows language distribution statistics

'insights age'

Shows age statistics

'translate Hello to Spanish'

Translates 'Hello' to Spanish

'help'

Shows available commands

'quit' or 'exit'

Exits the chatbot

Tips for Best Results:

• Commands are not case-sensitive - 'Patients' and 'patients' work the same

• You can ask questions naturally - the chatbot understands context

• Patient IDs are case-sensitive - use 'P001' not 'p001'

• Language names should be capitalized - 'Spanish' not 'spanish'

• Type 'help' anytime to see what the chatbot can do

18. Learning Outcomes

This project demonstrates many important programming concepts. Here's what you learn from this chatbot:

1. Data Structures Mastery

Lists:

• Creating and populating lists with data

• Using append() to add new items

• Using remove() to delete items

• Iterating through lists with for loops

• Using len() to count items

• List comprehension for filtering

• Checking if items exist with 'in' operator

Dictionaries:

• Creating dictionaries with key-value pairs

• Accessing values using keys

• Adding new key-value pairs

• Updating existing values

• Checking if keys exist

• Using dictionaries for fast lookups

• Nested dictionaries (dictionaries within dictionaries)

2. Real-World Programming Skills

• **Modular Design:** Breaking programs into smaller, reusable modules

• **Error Handling:** Managing errors gracefully when things go wrong

• **User Interface:** Creating friendly, conversational interfaces

• **Data Analysis:** Calculating statistics and generating reports

• **API Integration:** Working with external services like Google Translate

• **Testing:** Writing tests to ensure code works correctly

3. Healthcare IT Concepts

• Managing patient databases securely

• Handling multilingual healthcare communication

• Analyzing demographic data for healthcare planning

• Privacy and data organization in medical settings

• Clinical decision support systems

20. Summary and Conclusion

The ITEC5025 Week 5 repository showcases a sophisticated yet understandable chatbot system that demonstrates the power of Python's data structures.

Key Takeaways:

Lists are powerful for ordered collections:

Perfect for storing multiple items that you need to iterate through or count

Dictionaries enable fast lookups:

Essential when you need to find specific items quickly using keys

Combining data structures is effective:

Using both lists and dictionaries together gives you the best of both worlds

Real-world applications are complex:

Even a 'simple' chatbot requires thoughtful design and multiple technologies

Code organization matters:

Breaking code into modules makes it easier to understand, maintain, and expand

The Complete Picture:

This chatbot brings together:

• Core programming (Python, data structures, functions)

• Database management (storing and retrieving patient records)

• External APIs (Google Translate for multilingual support)

• Artificial Intelligence (NLTK and TensorFlow for understanding language)

• Statistical analysis (calculating demographics and trends)

• User experience (conversational interface, helpful responses)

Final Thoughts:

This project demonstrates that programming is not just about writing code - it's about solving real problems. The Hypotify chatbot solves a genuine healthcare challenge: managing patient information efficiently while supporting multiple languages.

By understanding lists and dictionaries through this practical example, you've learned fundamental concepts that apply to countless other programming projects. Whether you're building a website, analyzing data, or creating games, these data structures will be essential tools in your programming toolkit.

Congratulations on completing this comprehensive guide!