Matthew Johnson

November 14, 2023

IT FDN 110 A

Assignment05

GitHub Link: https://github.com/jwehttam/IntroToProg-Python-Mod05

Python: Advanced Collections and Error Handling

Introduction

This paper will explore the use of JSON (JavaScript Object Notation), the use of dictionaries in Python, and the basics of exceptions. These concepts are practical tools for programming, as you'll see in the accompanying code snippets from our course registration program.

What is JSON?

JSON, short for JavaScript Object Notation, is a text only data format that is easy to read and write. It is also easy for software to parse and generate. JSON can be written in any programming language, including Python.

JSON Structure and Syntax

JSON is built on two structures: Key/Value pairs and an Ordered List of Values.

A collection of key/value pairs - where each key is associated with a value. A value in a key/pair can be a number, a string, a list, a tuple, or even another dictionary. It should be noted that a key in the key/value pair is immutable, meaning that a key cannot be changed. In Python, a dictionary uses curly braces { } to define it.

An Ordered List of Values - a JSON object looks something like a dictionary in Python with key/value pairs. In our student registration program, we use JSON to store student information. Each student entry is a dictionary with keys like 'FirstName', 'LastName', and 'Course' (Figure 1).

```
# JSON
[
    {
        'FirstName': 'Matthew',
        'LastName': 'Johnson',
        'Course': 'Python 100'
    }
]
```

Figure 1: JSON Example

Reading and Writing in JSON

Python offers support for JSON through its 'json' library. You can convert a Python dictionary to a JSON string using 'json.dumps()', or save it to a file with 'json.dump()'. You can convert a JSON string to a dictionary using 'json.loads()', or read JSON from a file using 'json.load(). In our student registration program we use 'json.load()' to read the student data from our JSON formatted file (Figure 2).

```python
# json.load() example
file: io.TextIOWrapper = open(FILE_NAME, 'r')
students = json.load(file)
```

Figure 2: A json.load() Example

Dictionaries in Python

Dictionaries are an essential data structure for programming in Python. They are used to store collections of key/value pairs. Python dictionaries are denoted by curly braces '{ }' and keys are separated from values using a colon ' : '.

Dictionary Basics and Operations

In a really basic way, I think of a dictionary as a big box where you can put pairs of keys and values so you can retrieve them later on. As mentioned above, you use curly braces to create a dictionary. Here is an example from my program (Figure 3).

```python
# Create a new dictionary
student_data = {'FirstName': student_first_name, 'LastName': student_last_name,
'CourseName': course_name}
```

Figure 3: Dictionary Example

In this example, the dictionary is called 'student_data', and it has three key/value pairs:

- FirstName, LastName and CourseName are the key
- student_first_name, student_last_name and course_name are the values

As you can see, dictionaries are like customizable storage units for data, where each piece of data can be quickly accessed using a unique key. Dictionaries are *fairly easy for beginners to grasp and be used in their code.

Accessing and Modifying Dictionary Elements

Let's look at some ways of accessing and modifying elements in a dictionary. This is an important concept, being a beginner. I think of it as the base that I will build on top of as our program gets more complex going forward.

In Figure 3 above, we created a dictionary of 'student_data'. How can we access values from a dictionary using keys? If we wanted to print out the student's first name, we can do that as seen in (Figure 4). Unlike lists, which are indexed by numbers, we use a key instead.

```python
# Accessing Dictionary Elements
print(student_data['FirstName'])  # Output: Matthew (student_first_name 'value')
```

Figure 4: Accessing a Dictionary Element

If we want to modify a key's value in the dictionary, we could do that as seen in (Figure 5).

```python
# Modify a Key's Value
student_data['CourseName'] = 'Python 200'
```

Figure 5: Modify a Key's Value

Exceptions in Python

Exceptions in Python allow a programmer to interrupt the normal flow of their program to let them know that an error or unexpected condition has happened. Understanding errors and handling these exceptions helps a programmer with how to manage these disruptions and communicate them to the user.

Understanding Python Exceptions

An exception is a way of saying, "Hey, something unexpected has happened!" Understanding and handling exceptions help your program from crashing and can provide user friendly error messages to the end user.

For example, we learned how to handle a 'ValueError' in the student registration program. When a user is registering a student, we only want to accept that the first and last names contain only letters. If the user input doesn't meet that criteria, Python raises a 'ValueError' (Figure 6).

```
# ValueError Example
try:
    student_first_name = input('Enter the student\'s first name: ')
    if not student_first_name.isalpha():
        raise ValueError('The first name should only contain letters.')

# Output: Input error: The first name should only contain letters.
```

Figure 6: Raising a ValueError

By catching and handling this error, we will prompt the user to only enter a name consisting only of letters, without ending the program.

Custom Exception Handling

Custom error handling is about anticipating a potential problem and specifying how we can handle them. In our student registration program, we learned how to use 'try-except' blocks. When a block of code under 'try:' raises an exception, the program jumps down to the 'except' block, and executes that code instead (Figure 7).

```
# Try-Except Block Example
try:
    # Code that might raise an exception
except KeyboardInterrupt:
    # Handle Error
    print(f'\nInput canceled by user.')

# Output:
  What would you like to do: ^C
  Input canceled by user.
  Program Ended
```

Figure 7: Keyboard Interrupt Error

The program gracefully handles a 'KeyboardInterrupt' exception, which occurred when the user manually interrupted the execution of the program, like hitting 'Ctrl+C'. This can make the program more user friendly against other runtime errors.

Summary

In conclusion, advanced data collection and error handling are important in Python. From the organized structure of JSON to the flexibility of dictionaries and the strength of exception handling, these elements have a big role in creating efficient and error resistant code. As I worked on this week's assignment for the student registration program, adding these concepts has helped me by increasing the complexity of my program.