Discussion-2: Understanding Variables and Data Types

Discussion Topic:

Select only one of the following questions and offer a good, substantive reply. Some repetition is inevitable, but please aim to reply to a question that has not already been answered. Please include the question number you respond to, so we all can better understand your post, such as “#1”, “#2”, etc.:

1. Suppose you are building a Python application for storing the names, birth dates, and addresses of members of an extended family. Describe the strategy that you would use in order to determine the data types you would need for your application. Provide four data elements and the corresponding data types that you would use in your application.
2. How many data types do we have in Python? What IS a data type?
3. When accepting data from the user, the handy input() function in Python returns a STRING value. How can we transform this STRING value into an INTEGER value; or into a FLOAT value? Why would such transformation be necessary? How would you do it in Python code?
4. An element in a dictionary has two parts. What are they called? How are they used? Include an actual Python code sample.
5. The list and dictionary object types are two of the most important and often used types in a Python program. Compare the functionalities of those two objects. What are some ways to insert, update, and remove elements from lists and dictionaries? Why would you choose one data type over another? Include actual Python code samples.

Variables and data types are exceedingly essential concepts in computer programming, and we will explore them in good detail this week. We also cover our first data structure, Python's dictionaries. Data structures have enormous importance in computer science, so this week's study of dictionaries will lay the foundation for understanding these collections of data, the relationships among its members, and the operations that can be performed over them.

My Post:

Hello class,
For this discussion, I have chosen question #1, where you take the role of a software developer building a Python application for storing the names, birth dates, and addresses of members of an extended family. The question is divided into two parts. The first part asks to describe the strategy that you would use in order to determine the data types you would need for your application, while the second part asks to provide four data elements and the corresponding data types that you would use in your application.

A quick explanation of Python data types.

First, Python offers a vast array of data types. The Python documentation provides detailed descriptions of each data type, and you can find the list at the following link: Data TypesLinks to an external site..
“Python also provides some built-in data types, in particular, dictLinks to an external site., listLinks to an external site., setLinks to an external site. and frozensetLinks to an external site., and tupleLinks to an external site.. The strLinks to an external site. class is used to hold Unicode strings, and the bytesLinks to

an external site. and bytearrayLinks to an external site. classes are used to hold binary data" (Python Software Foundation (a), n.d., Data Type). Built-in data types in Python are fundamental data structures that come standard with Python; you don't need to import any external library to use them.

The table below shows Python's common data types.

Table-1

Common Data Types

Type	Notes
int	Numeric type: Used for variable-width integers.
float	Numeric type: Used for floating-point numbers.
string	Sequence type: Used for text.
list	Sequence type: A mutable container with ordered elements.
tuple	Sequence type: An immutable container with ordered elements.
dict	Mapping type: A container with key-values associated elements.

Note: from *Programming in Python 3,* by Bailey, 2016.

Strategy for Determining Data Types

To determine the data types needed for an application, it is crucial to analyze the data that needs to be collected and understand the application's functionality requirements. In general, this equates to these four key steps:

1. Identifying the Data: identifying what types of data the application will collect and handle, such as textual information and numerical data.
2. Understanding Data Operations: which operations will be performed on the data, such as sorting, searching, or complex manipulations, to ensure the chosen data types can support these functionalities.
3. Structuring Data Relations: how different pieces of data relate to each other and deciding on the appropriate structures (e.g., nested dictionaries or lists) to represent these relationships efficiently.
4. Planning for Scalability and Maintenance: future expansions or modifications to the application and selecting data types and structures that allow for modification, updates, and scalability.

For this specific application, this translates to the following steps:
Note that the information provided does not explicitly state whether the data needs to be manipulated (sorted or modified). However, for the application to be useful and functional, the data needs to be manipulated to some extent.

Based on the information provided, the application functionality requirements are as follows:

1. Storing Personal Information: storing basic personal information for each family member, such as names and birth dates.

2. Address Management: manage and store current and possibly multiple addresses for each family member.
3. Relationship Tracking: tracking and representing the relationships between different family members (e.g., parent-child, spouses, siblings).
4. Data Manipulation: functionalities for editing, sorting, and updating the stored information, including personal details, addresses, and family relationships.

Based on the information provided, the data that needs to be collected is as follows:

1. Names: this includes names and family members' names are text data
2. Birth dates: birth dates can be text data, numbers data, or a mix of both.
3. Address: addresses can be complex, potentially requiring storage of multiple addresses per family member with components like street, city, state, and zip code. It is a mix of numbers and text data.
4. Relationship: relationships between family members (e.g., parent-child, spouses, siblings) is text data.

Four data elements and the corresponding data types

Taking into account the application functionality requirements and data information the following are the four data elements and the corresponding data types.

1. Names: the string data type **str.** This allows us to easily store and manipulate individual names. I will use a **tuple** to separate the first name and last name, **name = ('first_name', 'last_name').** Tuples are great in this case because they are immutable, meaning once a tuple is created, it cannot be altered ensuring that the integrity of first and last names is preserved. Additionally, they are indexed meaning that they can be searched by index. For example, a list name tuple can be searched by last or first name. Furthermore, a tuple takes less space in memory than a dictionary or a list.
2. Birth Dates: they could technically be stored as strings, integers, lists, or dictionaries, however utilizing the **datetime.date** object from Python's **datetime** module has significant advantages such as easy date manipulations and functionality. For example, calculating ages, or sorting members by their birth dates. In most cases storing birth dates, requires converting input strings into **datetime.date** objects. Note that **datetime** is not a data type per se, is an object or a class.

   A **datetime.date** object utilizes the following data type :
   - **Year**: An integer representing the year, e.g., 2024.
   - **Month**: An integer representing the month, from 1 (January) to 12 (December).
   - **Day**: An integer representing the day of the month, from 1 to 31, depending on the month and year.

For example:
Note: the method date.fromisoformat() coverts strings into datetime.date() object with integer arguments.

```
from datetime import date
>>> date.fromisoformat('2019-12-04')
datetime.date(2019, 12, 4)
```

```
>>> date.fromisoformat('20191204')
datetime.date(2019, 12, 4)
>>> date.fromisoformat('2021-W01-1')
datetime.date(2021, 1, 4)
```

(Python Software Foundation (b), n.d., datetime - Basic date and time types)

3. Address: addresses have multiple components such as street, city, state, and zip code. I would use a dictionary data type **dict**. The dictionary key-value pair items structure is great for storing, modifying, and accessing the various parts of an address.
4. Relationship: relationships between family members, such as parent-child, spouses, and siblings. I would use a dictionary data type **dict** with embedded **List** and **tuple** data types. In this structure, the keys represent the types of relationships, and the values are lists of names or identifiers referencing other family members. This would allow for easy storing, modifying, and accessing relationships data.

Below is an example of what the Python code could be like:

```
from datetime import datetime

        user_123 = {
          "name": ("John", "Doe"),  # Using tuple for the name
          "birth_date": datetime(1974, 6, 5),  # Using datetime for birth dates
          "address": {  # Using a dictionary for the address
            "street": "123 My Street",
            "city": "Mytown",
            "state": "Mystate",
            "zip_code": "12345"
          },
          "relationships": {  # Using a dictionary with embedded lists and tuples
            "spouse": ("Jane", "Doe"),
            "children": [("George", "Doe"), ("Laura", "Doe")],
            "parents": [("Paul", "Doe"), ("Lucy", "Doe")],
          }
        }
```

-Alex

References:

Bailey, M. (2016, August). Chapter 3: Types, *Programming in Python 3.* Zyante Inc.

Python Software Foundation (a). (n.d.). Data Type. *Python*.
python.org. https://docs.python.org/3/library/datatypes.htmlLinks to an external site.

Python Software Foundation (b). (n.d.). datetime - Basic date and time types *Python*. python.org. https://docs.python.org/3/library/datetime.html