NESTED DATA

An Introduction to Computer Science

Let's learn about Nested Data

The Types

Primitive Types

- Integer
- Float
- Boolean
- String
- None

Composite Types

- List
- Dictionary

At this point, we've learned about 7 different types.
The 5 primitive types are Integer, Float, Boolean, String, and None.
The 2 composite types we've seen so far are Lists and Dictionaries.
These 2 new composite types are special because they are composed of other types.

The Types

Primitive Types

- Integer
- Float
- Boolean
- String
- None

Composite Types

- List
- Dictionary
- Tuple
- Sets
- Classes

There are actually many more composite types in Python.
In fact, there are even ways to create entirely new primitive and composite types.
But for now, we'll stick to these basic 7 types.

Types in Types

```
prices = [40, 60, 50]
```

List of integers

```
book_prices = {"Harry Potter": 4.50,
               "War & Peace": 8.75,
               "Good Omens": 12.50}
```

Dictionary of Strings to Floats

With lists, we learned that composite types had subtypes.
For example, you could have a List of Integers.
Dictionaries have multiple subtypes, one for each key and one for each value.
For example, you could have a dictionary that maps String Keys to Float values.

Nesting Types

- List of
  - List of
    - Dictionary mapping
      - Strings to
      - List of
        - Dictionary mapping
          - Integers to
          - List of
            - List of
              - Dictionary mapping
                - Strings to
                - List of
                - …

You can also use composite types as subtypes.
In other words, you can have lists of dictionaries or dictionaries of lists or even lists of lists.
There are no practical limits to how many times you can nest composite types.

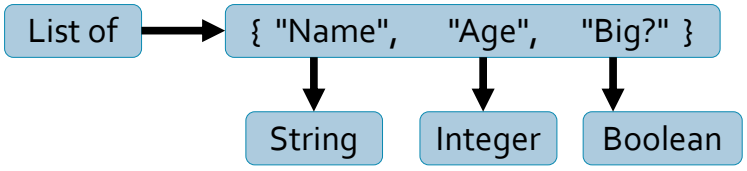
Nesting Literals

```
[
 {"Name": "Klaus",  "Age": 17, "Big?": True},
 {"Name": "Tigger", "Age": 12, "Big?": True},
 {"Name": "Wrex",   "Age": 2,  "Big?": False}
]
```

Let us look at a concrete example.
Shown here is a list of dictionaries.
Each dictionary is being used as a Record with the same keys; these dictionaries can therefore represent a bunch of animals.
The subtype of the list is dictionary.
The keys of those dictionaries are all strings, and the values are either strings or integers.

A Representative Element

```
[
 {"Name": "Klaus",   "Age": 17, "Big?": True},
 {"Name": "Tigger",  "Age": 12, "Big?": True},
 {"Name": "Wrex",    "Age": 2,  "Big?": False}
]
```



To quickly understand the structure of a complex nested structure, we find it useful to model the memory.
In the diagram below, we see that we have a list of dictionaries, where the named keys map to different types.

Processing Nested Data

```
animals = [
 {"Name": "Klaus",  "Age": 17, "Big?": True},
 {"Name": "Tigger", "Age": 12, "Big?": True},
 {"Name": "Wrex",   "Age": 2,  "Big?": False}
]


for animal in animals:
   print(animal["Name"])
```

1) process as list

2) process individual dictionary

As you process this complex nested data's structure, you have to stay aware of where you are.
This is similar to needing to navigate a maze.
Consider a list of dictionaries like the one shown before.
If we wanted to print the name of each animal, we would first need to process them a list using a For loop.
Then, we can process an individual dictionary.

Nested Dictionaries

```
event = {
    "Name": "Solar Eclipse Party",
    "Attendees": 403,
    "Date": {
        "Month": "April",
        "Day": 8,
        "Year": 2024
    }
}

event["Date"]["Month"]
```

Nested dictionaries

Chained dictionary access

Here we see a dictionary that is composed of dictionaries.
When we access these nested dictionaries, we use square brackets chained together.
The expressions on the right access various elements of the dictionary.
Nesting dictionaries is an excellent way to cluster information.

Layers of Nesting

```
events = [
    { "Name": "Python Class",
      "Date": {
        "Day": "Thursday",
        "Start": 9,
        "End": 11}},
    { "Name": "Dinner Date",
      "Date": {
        "Day": "Friday",
        "Start": 6,
        "End": 8}},
    { "Name": "Doctor Appointment",
      "Date": {
        "Day": "Monday",
        "Start": 11,
        "End": 12
      }}
]
```

```
>>> events[2]["Name"]
'Doctor Appointment'

>>> events[0]["Date"]
{'Day': 'Thursday', 'End': 11, 'Start': 9}

>>> events[1]["Date"]["Start"]
6
```

As previously mentioned, there are no limits to how much nesting can happen.
Here we see a list of dictionaries of dictionaries, to represent an event calendar.
Instead of processing this list with a for loop, we can chain list indexing and dictionary access to lookup specific elements.
As our nested structures grow more complex, it becomes more and more important to understand the nature of the data to learn to navigate it.