1. [SOLUTIONS] Data Structures

People

You are to write a program which works with a collection of people. We are interested in name and age only of each person. What data structure would use to represent the collection?
Answer: This could be a dictionary with the name of the person as a key: **`{"John Smith": 25, "Jane Doe": 22}`**

RoomsInABuilding

The Sir Alwyn Williams Building has 5 floors. There are a number of rooms on each floor. Each room has a number, a capacity, whether it has a window, and whether or not it has a data projector in it. What data structure could hold this information?
Answer: This could be a dictionary with each floor corresponding to a list of rooms, each room represented by a dictionary:

```
{ 1: [
  {"number": 6, "capacity": 30, "window": True, "projector": False}
  {"number": 11, "capacity": 20, "window": False, "projector": True}
],
  2: [
  {"number": 14, "capacity": 30, "window": True, "projector": False}
  {"number": 7, "capacity": 20, "window": False, "projector": False}
] }
```

PhoneContactsList

In this exercise, we'll consider how contacts are stored on your mobile phone. We'd like to be able to find/lookup any phone numbers using the name of the person we'd like to call.

1. Consider the simplest case of a structure that can hold a single entry consisting of the persons name and an associated phone number. What data structure could you use to store the following entry 'Mick Jagger' with telephone number 07731 423321?

Answer: A dictionary could be used with the person's name as the key and the number as the associated value **`{"Mick Jagger": "07731 423321"}`*****. Note that the phone number would need to be stored as string to preserve the leading zero.***

2. For the phone numbers we'd like replace the single value with a more complex set of information which includes the type of phone: work, home, mobile, the number associated with it and the type of phone that is the default number for this contact. How would you adjust your data structure to include this additional information? For 'Mick

Jagger' 07731 423321 is his mobile number and his home phone is 0112 337 8932. Mick is abroad a lot so his default number should be his mobile phone.

Answer: The single number value could be replaced by a nested dictionary of phone types and default number values **`{“Mick Jagger”: {‘mobile’:’07731 423321’, ‘home’:’0112 337 8932, ‘default’:’mobile’’} }`**

3. We have the structure for one contact but most people have more than one friend or family member they'd like to call. How would you extend your data structure to add a new entry for 'Keith Richards', mobile number 07832 778412, home phone 0133 227 3345 and default number home phone?

Answer: Add another name to the main dictionary and another nested dictionary that contains their phone number information **`{“Mick Jagger”: {“mobile”:”07731 423321”, “home”:”0112 337 8932”, “default”:”mobile’’} , “Keith Richards”: {“mobile”: 07832 778412, “home”:”0133 227 3345”, “default”:”home”} }`**

4. Declare and initialise a variable called **`contacts`** to store the two contacts and then write code to lookup up and print the following values:
   a. Mick Jagger's mobile number
   b. Keith Richard's home phone
   c. Mick Jagger's default number

Answer:

```
print(contact_list[ “Mick Jagger” ][ “mobile” ])
print(contact_list[ “Keith Richards” ][ “home” ])
#The last lookup needs two seperate pieces of information from the
contact_list data structure. The value returned by the nested contact list
gives the key name for the default phone number type.
print (contact_list[ “Mick Jagger” ][ contact_list[“Mick Jagger”][“default”]
]
```

Music

A program is needed that works with music. We need a data structure to hold a tune. Let's consider what's in a tune?

- It consists of a sequence of notes played one at a time - one after the other.
- Each note has a pitch - how high or low the note sounds - and we will assume this is one of 56 notes.
  - These notes are divided into 8 octaves, numbered 1 to 8.
  - Each octave has 7 notes in sequence, letter A to G.
- A note may be a semi-tone higher (sharp) or a semi-tone lower (flat) or natural.
- A note has a duration - how long it is played for.
  - This is measured in beats - 1/16, ⅛, ¼, ½, 1, 2, 4
  - A note can be “dotted”, which increases the duration by half its main duration

1. Come up with a suitable Python data structure to hold a single note.

Answer: Each note is a dictionary with entries for octave (integer - 1-8), note letter (string "A"-"G"), natural/flat/sharp (integer - 0,1,2), duration (float), dotted (boolean) e.g. **`{ "octave" : 4, "note" : "B", "nfs" : 1, "duration" : 1.0, "dotted" : False }`** ***is B flat in the 4th octave, lasting for a single beat, not dotted.***

2. Extend this to hold a complete tune as described above.

Answer: A complete tune would be a list of dictionaries in this format.

3. How would you extend your data structure to include *chords* - multiple notes played at the same time?

Answer: Each item in the list, instead of being just the note dictionary, could be a list of note dictionaries.

TextDocument

In this exercise, we'll consider how documents are represented as a data structure.

1. Consider a plain text document as an ordered collection of paragraphs. What data structure would you use to hold it?

Answer: List of strings. Each string represents a paragraph - a sequence of characters that is implicitly terminated with a new line character. For example:

```
[ "This is the first paragraph of text. It is quite short.", "And this is the
second.  Even shorter." ]
```

2. Now imagine that each paragraph can have styling attached - for this example, consider only the styling bold, italics, underline, which can be applied in any order. Adjust/extend your data structure as necessary.

Answer: Add a second list, this time of dictionaries - one dictionary for each line. The dictionaries contain three entries with the keys "bold", "italic" and "underline". The values associated with these entries are all initially False. If the paragraph corresponding to a dictionary has been styled, then the values of the relevant styling entries will be set to True. For example, if the first paragraph in the example above was styled with bold and italic, and the second with nothing, this second list would be as follows:

```
[ { "bold" : True, "italic" : True, "underline" : False }, { "bold" : False,
"italic" : False, "underline" : False } ]
```

Alternatively, each string entry in the first list could be expanded to be a dictionary containing a "text" key with the text string as value, and a "styling" key with the corresponding styling dictionary as the value.

3. Now, individual sequences of characters within the text can be styled, using the same three options as in (2) above. Extend your data structure further.

Answer: A third list is introduced - each entry specifies the styling of fragments of the corresponding paragraph. The entry is itself a list - containing dictionaries. Each dictionary specifies the styling of one fragment in the paragraph. The dictionary contains entries for the start and end point of the fragment in the paragraph, and also a dictionary like the ones used in (b) above to specify the particular styling. For example, if the first word of the second paragraph was bold, but no other fragment was styled at all, then this list would be as follows:

```
[ [ { "start" : 0, "end" : 2, "styling" :  { "bold":True, "italic":False,
"underline":False } } ], [] ]
```

Alternatively, as in the alternate version of (2) above, the dictionary for a whole line (containing text and styling) can be extended with a "fragmentStyling" entry, the value for which is a list of fragments for that line, as in the list above.

4. Finally, how would you extend your data structure to enable the document to contain images inserted at arbitrary points in the text. For this example, it doesn't matter exactly how an image is represented - say it's an empty dictionary for now; the challenge is to work out how to mix text and images.

Answer: A fourth list of a similar structure as that in (3), apart from replacing "start" and "end" entries with a single "pos" entry - the image should be inserted before this position in the paragraph - and replacing the "styling" entry by an "image" entry. For example, if an image were inserted directly after the first word of the first paragraph, then this look as follows:

```
[ [ { "pos" : 4, "image" :  {} } ], [] ]
```

Alternatively, as before, the dictionary for a whole line (containing text, styling and fragment styling) can be extended with an "images" entry, the value for which is a list of the image dictionaries for that line, as specified above.