Introduction to Python

Reading / References

- Python Dictionaries[1]
- More docs on Python Dictionaries[2]
- Python Lists[3]
- Python Tuples[4]

Notes

In this section we will review some of the key Python data structures and describe suitable uses for them. This is not meant to be a complete introduction to Python, but only a refresher of some key ideas, with an emphasis on how information may be stored in Python and the consequences of different storage decisions.

Lists, Tuples and Dictionaries

Regardless of the programming language of choice, there are some common types used when representing data. They come in two flavors:

- Basic types for representing character *strings*, *numbers*, *dates* (some times), *boolean* values.
- Compound/Container types that can hold many values. The three most important compound types are *lists*, *tuples* and *dictionaries* (often also called *maps*). Understanding what each of these does and when it is suitable is an important distinction.

Lists Chances are you are already quite familiar with lists.

- A list contains zero or more elements in a sequence. They can contain an arbitrary number of elements.
- We can iterate over the elements of a list with a `for-in` loop. Or we can access a specific location if we know its index.
- A list can in theory contain arbitrary elements, though in practice all elements we put in a list will be of the same type (e.g. all numbers).
- We can create a list with the literal notation like `[2, 5, 6]` or starting with an empty list like `list()`.
- We can use indexing like `x[2]` to get at a value in the list (indices start at 0).

[1] https://docs.python.org/3/tutorial/datastructures.html#dictionaries
[2] https://docs.python.org/3/library/stdtypes.html#typesmapping
[3] https://docs.python.org/3/tutorial/datastructures.html
[4] https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences

- We can add elements to a list using append.

As a simple example, we could have a list containing the names of all teams in our conference (data obtained from http://www.heartlandconf.org/:

```
teams = ["HANOVER", "MOUNT␣ST.␣JOSEPH", "ROSE-HULMAN",
         "TRANSYLVANIA", "ANDERSON", "BLUFFTON",
         "FRANKLIN", "EARLHAM", "MANCHESTER", "DEFIANCE"]
teams[2]    # Rose-Hulman
teams[2:4] # A "slice" of entries 2 through 4

for team in teams:
    print(team)

# Creating a new empty list then copying the elements over
# This is meant as an example. Do NOT copy lists this way.
# Use teams.copy() instead.
teamsCopy = list()
for team in teams:
    teamsCopy.append(team)
```

Tuples May not have used tuples before, but they are quite useful in putting together heterogeneous but coupled elements.

- A tuple is a sequence of values that has a *fixed* number of elements, often called its *arity*.
- The elements in the tuple can be of any type.
- A tuple is "immutable". We cannot add elements to it or change existing elements.
- Tuples are formed by using parentheses around a set of values separated by commas. For instance: t=("HANOVER", 14, 4)
- We can access an element in a tuple using index notation: t[0].
- We can also assign all tuple values to variables at once using multiple assignment, as in the example below.

```
t = ("HANOVER", 14, 4)
t[0]    # <-- "HANOVER"
(name, wins, losses) = t  # multiple assignment
name, wins, losses = t    # also works
```

As a longer example, we could combine lists and tuples. For example we may have a list of tuples, and process it in a for-in loop with multiple assignment:

```
results = [
    ("HANOVER", 14, 4), ("MOUNT␣ST.␣JOSEPH", 14, 4), ("ROSE-HULMAN", 14, 4),
    ("TRANSYLVANIA", 12, 6), ("ANDERSON", 9, 9), ("BLUFFTON", 8, 10),
    ("FRANKLIN", 7, 11), ("EARLHAM", 6, 12), ("MANCHESTER", 4, 14),
    ("DEFIANCE", 2, 16)
]

for team, wins, losses in results:
    print("%s:␣%d␣Wins,␣%d␣Losses" % (team, wins, losses))
```

```
# Alternative:
for tuple in results:
    print("%s: %d Wins, %d Losses" % tuple)
```

Notice the expression `"%s: %d Wins, %d Losses" % tuple` in the contents of the print statement. This is the use of the percent operator for doing *string formating*. It has the general form:

```
string % values
```

Where `values` is a tuple of values whose arity matches the placeholders in the string. Read more about it and other formats in the input/output[5] section of the Python tutorial.

Dictionaries/Maps A *dictionary*, often referred to also as a *map*, is a structure that associates "keys" with "values".

- Each key has a unique value corresponding to it.
- Any immutable type can be used as a key. It is common to use strings.
- You can create a dictionary via its literal expression: `{ key1: value1, key2: value2 }`
- You can create an empty dictionary with `dict()`.
- Elements can be accessed by key indexing, like so: `x["foo"]`.
- You can also iterate over the keys and values.
- You can check for the presence of a key in a dictionary with the form: `key in dict`
- You can update the value for a given key or add a new key to a dictionary.
- Typically the values in a dictionary are of the same type, but that is not a requirement.

For example, we used a tuple earlier to represent the information of a team and their win and loss record. We could instead have used a dictionary:

```
team = { "name": "HANOVER", "wins": 14, "losses": 4 }
team["name"]       # <-- "HANOVER"

# Iterate over the keys:
for key in team:
    print(key, team[key])

# Iterate over key-value pairs:
for key, value in team.items():
    print(key, value)
```

As an example, we could store the teams we worked with earlier in a dictionary, indexed by the team names:

```
results = {
    "HANOVER": ("HANOVER", 14, 4),
    "MOUNT ST. JOSEPH": ("MOUNT ST. JOSEPH", 14, 4),
    "ROSE-HULMAN": ("ROSE-HULMAN", 14, 4),
    "TRANSYLVANIA": ("TRANSYLVANIA", 12, 6),
```

[5] https://docs.python.org/3.1/tutorial/inputoutput.html#fancier-output-formatting

```
    "ANDERSON": ("ANDERSON", 9, 9),
    "BLUFFTON": ("BLUFFTON", 8, 10),
    "FRANKLIN": ("FRANKLIN", 7, 11),
    "EARLHAM": ("EARLHAM", 6, 12),
    "MANCHESTER": ("MANCHESTER", 4, 14),
    "DEFIANCE": ("DEFIANCE", 2, 16)
}
results["HANOVER"]
```

An example

As a motivating example, we will consider the following problem: We have stored the text from a certain book in a text file. We will use for our example a transcript of the *Tale of Two Cities* by Charles Dickens, which can be found in this file[6] provided by the "E-Texts" website[7]. We would like to process it in a way that would facilitate answering some questions. For example:

- How many words and how many paragraphs are there in the text?
- What words are the most frequent and how often do they appear?
- Same question but for each chapter.
- What are all the places where a specific word appears in the text?
- How often does a specific pair of words appear in the same paragraph?
- Which pairs of words appear most often in the same paragraph?

Activity: Discuss how you would represent the text of this book in Python to facilitate answering these questions, using the data structures described above. Present at least three different approaches and discuss advantages and disadvantages of each. Some approaches might make some of the questions easier but other questions harder.

Practice

1. You will store courses that you have taken, identifying the term that you took them on the course prefix (e.g. CS) and your grade as a gpa number. Decide how you would represent this in Python and then make a small list of such courses including at least two different terms and two different prefixes. You can of course make your grades up if you prefer. Store these in a variable called courses.
2. Write code that would compute your overall gpa. You need to keep track of a running total as well a number of courses, in order to compute an average at the end.
3. Write code that would compute your gpa for each of the prefixes. You may want to use a dictionary or two to store the results.
4. Read up on how to sort a list of items in Python, and sort your courses based on grade, starting from the highest.

[6] http://www.textfiles.com/etext/FICTION/2city10.txt
[7] http://www.textfiles.com/etext/