A Lisp Programmer Living in Python-Land: The Hy Programming Language

Mark Watson

A Lisp Programmer Living in Python-Land: The Hy Programming Language

Mark Watson

This book is for sale at http://leanpub.com/hy-lisp-python

This version was published on 2020-10-03

Contents

Preface

While this is a book on the Hy Lisp language, we have a wider theme here. In an age where artificial intelligence (AI) is a driver of the largest corporations and government agencies, the question is how do individuals and small organizations take advantage of AI technologies given disadvantages of small scale. The material I chose to write about here is selected to help you, dear reader, survive as a healthy small fish in a big bond.

I have been using Lisp languages professionally since 1982 and have written books covering the Common Lisp and Scheme languages. Most of my career has involved working on AI projects so tools for developing AI applications will be a major theme. In addition to covering the Hy language, you will get experience with AI tools and techniques that will help you craft your own AI platforms regardless of whether you are a consultant, work at a startup, or a corporation.

This book covers many programming topics using the Lisp language **Hy** that compiles to Python AST and is compatible with code, libraries, and frameworks written in Python. The main topics we will cover and write example applications for are:

- Relational and graph databases
- Web app development
- Web scraping
- Accessing semantic web and linked data sources like Wikipedia, DBpedia, and Wikidata
- Automatically constructing Knowledge Graphs from text documents, semantic web and linked data
- Deep Learning
- Natural Language Processing (NLP) using Deep Learning

The topics were chosen because of my work experience and the theme of this book is how to increase programmer productivity and happiness using a Lisp language in a bottom-up development style. This style relies heavily on the use of an interactive REPL for exploring APIs and writing new code. I chose the above topics based on my experience working as a developer and researcher. Please note: you will see the term REPL frequently in this book. REPL stands for *Read Eval Print Loop*.

Some of the examples are very simple (e.g., the web app examples) while some are more complex (e.g., Deep Learning and knowledge graph examples). Regardless of the simplicity or complexity of the examples I hope that you find the code interesting, useful in your projects, and fun to experiment with.

Setting Up Your Development Environment

This is a hands-on book! I expect you, dear reader, to follow along with the examples as you read this book. I assume that you know some Python and know how to use the command line tools **python** and **pip** and use a virtual Python environment like Anaconda (**conda**)[1] or **virtualenv**[2]. Personally I prefer **conda** but you can use any Python 3.x setup you like as long as you have a few packages installed.

You can install the current stable version of **Hy** using:

```
pip install git+https://github.com/hylang/hy.git
```

Depending on which examples you run and experiment with you will also need to install some of the following libraries:

```
pip install beautifulsoup4 Flask Jinja2 Keras psycopg2
pip install rdflib rdflib-sqlite spacy tensorflow
pip install PyInquirer
```

The Hy language is under active development and it is not unusual for libraries and frameworks created more than a few months before the current Hy release to break. As a result of this, I have been careful in the selection of book material to leave out interesting functionality and libraries from the Hy ecosystem that I feel might not work with new releases. Here we stick with a few popular Python libraries like Keras, TensorFlow, and spaCy and otherwise we will work with mostly pure Hy language code in the examples.

What is Lisp Programming Style?

I will give some examples here and also show exploratory Hy language REPL examples later in the book. How often do you search the web for documentation on how to use a library, write some code only to discover later that you didn't use the API correctly? I reduce the amount of time that I spend writing code by having a Lisp REPL open so that I can experiment with API calls and returned results while reading the documentation.

When I am working on new code or a new algorithm I like to have a Lisp REPL open and try short snippets of code to get working code for solving low level problems, building up to more complex code. As I figure out how to do things I enter code that works and that I want to keep in a text editor and then convert this code into my own library. I then iterate on loading my new library into a REPL and stress test it, look for API improvements, etc.

[1]https://www.anaconda.com/
[2]https://virtualenv.pypa.io/en/latest/

I find, in general, that a "bottom-up" approach gets me to working high quality systems faster than spending too much time doing up front planning and design. The problem with spending too much up front time on design is that we change our minds as to what makes the most sense to solve a problem as we experiment with code. I try to avoid up front time spent on work that I will have to re-work or even toss out.

Hy is Python, But With a Lisp Syntax

When I need a library for a Hy project I search for Python libraries and either write a thin Hy language "wrapper" around the Python library or just call the Python APIs directly from Hy code. You will see many examples of both approaches in this book.

How This Book Reflects My Views on Artificial Intelligence and the Future of Society and Technology

Since starting work on AI in 1982 I have seen the field progress from a niche technology where even international conferences had small attendances to a field that is generally viewed as transformative. In the USA there is legitimate concern that economic adversaries like China will exceed our abilities to develop core AI technologies and integrate these technologies into commercial and military systems. As I write this in February 2020, some people in our field including myself believe that the Chinese company Baidu may have already passed Google and Microsoft in applied AI.

Even though most of my professional work in the last five years has been in Deep Learning (and before that I worked with the Knowledge Graph at Google on a knowledge representation problem and application), I believe that human level Artificial General Intelligence (AGI) will use hybrid Deep Learning, "old fashioned" symbolic AI, and techniques that we have yet to discover.

This belief that Deep Learning will not get us to AGI capabilities is a motivation for me to use the Hy language because it offers transparent access to Python Deep Learning frameworks with a bottom-up Lisp development style that I have used for decades using symbolic AI and knowledge representation.

I hope you find that Hy meets your needs as it does my own.

About the Book Cover

The official Hy Language logo is an octopus:

The Hy Language logo Cuddles by Karen Rustad

Usually I use photographs that I take myself for covers of my LeanPub books. Although I have SCUBA dived since I was 13 years old, sadly I have no pictures of an octopus that I have taken myself. I did find a public domain picture I liked (that is the cover of this book) on Wikimedia. **Cover Credit**: Thanks to Wikimedia user Pseudopanax for placing the cover image in the public domain.

A Request from the Author

I spent time writing this book to help you, dear reader. I release this book under the Creative Commons "share and share alike, no modifications, no commercial reuse" license and set the minimum purchase price to $5.00 in order to reach the most readers. Under this license you can share a PDF version of this book with your friends and coworkers. If you found this book on the web (or it was given to you) and if it provides value to you then please consider doing one of the following to support my future writing efforts and also to support future updates to this book:

- Purchase a copy of this book or any other of my leanpub books at https://leanpub.com/u/markwatson[3]
- Hire me as a consultant[4]

I enjoy writing and your support helps me write new editions and updates for my books and to develop new book projects. Thank you!

Acknowledgements

I thank my wife Carol for editing this manuscript, finding typos, and suggesting improvements.

I would like to thank Pascal (Reddit user chuchana) for corrections and suggestions. I would like to thank Carlos Ungil for catching a typo and reporting it. I would like to thank Jud Taylor for finding several typo errors.

[3]https://leanpub.com/u/markwatson
[4]https://markwatson.com/

Introduction to the Hy Language

The Hy programming language[5] is a Lisp language that inter-operates smoothly with Python. We start with a few interactive examples that I encourage you to experiment with as you read. Then we will look at Hy data types and commonly used built-in functions that are used in the remainder of this book.

I assume that you know at least a little Python and more importantly the Python ecosystem and general tools like **pip**.

Please start by installing Hy in your current Python environment:

```
pip install git+https://github.com/hylang/hy.git
```

We Will Often Use the Contributed let Macro in Book Example Code

In Scheme, Clojure, and Common Lisp languages the **let** special form is used to define blocks of code with local variables and functions. I will require (or import) the contributed **let** macro, that substitutes for a built-in special form in most examples in this book, but I might not include the **require** in short code listings. Always assume that the following lines start each example:

```
1  #!/usr/bin/env hy
2
3  (require [hy.contrib.walk [let]])
```

Line 1 is similar to how we make Python scripts into runnable programs. Here we run **hy** instead of **python**. Line 3 imports the **let** macro. We will occasionally use **let** for code blocks with local variable and function definitions and also for using closures (I will cover closures at the end of this chapter):

[5]http://docs.hylang.org/en/stable/

```
#!/usr/bin/env hy

(require [hy.contrib.walk [let]])

(let [x 1]
  (print x)
  (let [x 33]
    (print x)
    (setv x 44)
    (print x))
  (print x))
```

The output is:

```
1
33
44
1
```

Notice that setting a new value for **x** in the inner **let** expression does not change the value bound to the variable **x** in the outer **let** expression.

Using Python Libraries

Using Python libraries like TensorFlow, Keras, BeautifulSoup, etc. are the reason I use the Hy language. Importing Python code and libraries and calling out to Python is simple and here we look at sufficient examples so that you will understand example code that we will look at later.

For example, in the chapter **Responsible Web Scraping** we will use the BeautifulSoup library. We will look at some Python code snippets and the corresponding Hy language versions of these snippets. Let's first look at a Python example that we will then convert to Hy:

```
from bs4 import BeautifulSoup

raw_data = '<html><body><a href="http://markwatson.com">Mark</a></body></html>'
soup = BeautifulSoup(raw_data)
a_tags = soup.find_all("a")
print("a tags:", a_tags)
```

In the following listing notice how we import other code and libraries in Hy. The special form **setv** is used to define variables in a local context. Since the **setv** statements in lines 3, 5, and 6 are used at the top level, they are global in the Python/Hy module named after the root name of the source file.

```
1  $ hy
2  hy 0.18.0 using CPython(default) 3.7.4 on Darwin
3  => (import [bs4 [BeautifulSoup]])
4  => (setv raw-data "<html><body><a href=\"http://markwatson.com\">Mark</a></body></ht\
5  ml>")
6  => (setv soup (BeautifulSoup raw-data "lxml"))
7  => (setv a (.find-all soup "a"))
8  => (print "atags:" a)
9  atags: [<a href="http://markwatson.com">Mark</a>]
10 => (type a)
11 <class 'bs4.element.ResultSet'>
12 => (dir a)
13 ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dict__', '\
14 __dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getattribut\
15 e__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__in\
16 it_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__module__', '__mul__', \
17 '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__r\
18 mul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', '\
19 __weakref__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop'\
20 , 'remove', 'reverse', 'sort', 'source']
```

Notice in lines 3 and 6 that we can have "-" characters inside of variable and function names (**raw-data** and **find-all** in this case) in the Hy language where we might use "_" underscore characters in Python. Like Python, we can use **type** get get the type of a value and **dir** to see what symbols are available for a object.

Global vs. Local Variables

Although I don't generally recommend it, sometimes it is convenient to export local variables defined with **setv** or in a **let** macro expansion to be global variables in the context of the current module (that is defined by the current source file). As an example:

```
Marks-MacBook:deeplearning $ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (defn foo []
... (global x)
... (setv x 1)
... (print x))
=> (foo)
1
=> x
```

```
1
=>
```

Before executing function **foo** the global variable **x** is undefined (unless you coincidentally already defined somewhere else). When function **foo** is called, a global variable **x** is defined and then it equal to the value 1.

Using Python Code in Hy Programs

If there is a Python source file, named for example, *test.py* in the same directory as a Hy language file:

```
def factorial (n):
  if n < 2:
    return 1
  return n * factorial(n - 1)
```

This code will be in a module named **test** because that is the root source code file name. We might import the Python code using the following in Python:

```
import test

print(test.factorial(5))
```

and we can use the following in Hy to import the Python module **test** (defined in *test.py*):

```
(import test)

(print (test.factorial 5))
```

Running this interactively in Hy:

```
$ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import test)
=> test
<module 'test' from '/Users/markw/GITHUB/hy-lisp-python/test.py'>
=> (print (test.factorial 5))
120
```

If we only wanted to import **BeautifulSoup** from the Python BeautifulSoup library **bs4** we can specify this in the **import** form:

```
(import [bs4 [BeautifulSoup]])
```

Using Hy Libraries in Python Programs

There is nothing special about importing and using Hy library code or your own Hy scripts in Python programs. The directory **hy-lisp-python/use_hy_in_python** in the git repository for this book https://github.com/mark-watson/hy-lisp-python[6] contains an example Hy script **get_web_-page.hy** that is a slightly modified version of code we will explain and use in the later chapter on web scraping and a short Python script **use_hy_stuff.py** that uses a function defined in Hy:

get_web_page.hy:

```
 1  (import argparse os)
 2  (import [urllib.request [Request urlopen]])
 3
 4  (defn get-raw-data-from-web [aUri &optional [anAgent
 5                                               {"User-Agent" "HyLangBook/1.0"}]]
 6    (setv req (Request aUri :headers anAgent))
 7    (setv httpResponse (urlopen req))
 8    (setv data (.read httpResponse))
 9    data)
10
11  (defn main_hy []
12    (print (get-raw-data-from-web "http://markwatson.com")))
```

We define two functions here. Notice the optional argument **anAgent** defined in lines 4-5 where we provide a default value in case the calling code does not provide a value. In the next Python listing we import the file in the last listing and call the Hy function **main** on line 4 using the Python calling syntax.

Hy is the same as Python once it is compiled to an abstract syntax tree (AST).

hy-lisp-python/use_in_python:

```
1  import hy
2  from get_web_page import main_hy
3
4  main_hy()
```

What I want you to understand and develop a feeling for is that Hy and Python are really the same but with a different syntax and that both languages can easily be used side by side.

[6]https://github.com/mark-watson/hy-lisp-python

Replacing the Python slice (cut) Notation with the Hy Functional Form

In Python we use a special notation for extracting sub-sequences from lists or strings:

```
$ python
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
>>> s = '0123456789'
>>> s[2:4]
'23'
>>> s[-4:]
'6789'
>>> s[-4:-1]
'678'
>>>
```

In Hy this would be:

```
$ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (setv s "0123456789")
=> (cut s 2 4)
'23'
=> (cut s -4)
'6789'
=> (cut s -4 -1)
'678'
=>
```

It also works to use **cut** with **setv** to destructively change a list; for example:

```
=> (setv x [0 1 2 3 4 5 6 7 8])
=> x
[0, 1, 2, 3, 4, 5, 6, 7, 8]
=> (cut x 2 4)
[2, 3]
=> (setv (cut x 2 4) [22 33])
=> x
[0, 1, 22, 33, 4, 5, 6, 7, 8]
```

Iterating Through a List With Index of Each Element

We will use **lfor** as a form of Python list comprehension; for example:

```
1  => (setv sentence "The ball rolled")
2  => (lfor i (enumerate sentence) i)
3  [(0, 'T'), (1, 'h'), (2, 'e'), (3, ' '), (4, 'b'), (5, 'a'), (6, 'l'), (7, 'l'), (8,\
4   ' '), (9, 'r'), (10, 'o'), (11, 'l'), (12, 'l'), (13, 'e'), (14, 'd')]
5  => (setv vv (lfor i (enumerate sentence) i))
6  => vv
7  [(0, 'T'), (1, 'h'), (2, 'e'), (3, ' '), (4, 'b'), (5, 'a'), (6, 'l'), (7, 'l'), (8,\
8   ' '), (9, 'r'), (10, 'o'), (11, 'l'), (12, 'l'), (13, 'e'), (14, 'd')]
9  => (for [[a b] vv]
10 ... (print a b))
11 0 T
12 1 h
13 2 e
14 3
15 4 b
16 5 a
17 6 l
18 7 l
19 8
20 9 r
21 10 o
22 11 l
23 12 l
24 13 e
25 14 d
26 =>
```

On line 2, the expression **(enumerate sentence)** generates one character at a time from a string. **enumerate** operating on a list will generate one list element at a time.

Line 9 shows an example of *destructuring*: the values in the list **vv** are tuples (tuples are like lists but are immutable, that is, once a tuple is constructed the values it holds can not be changed) with two values. The values in each tuple are copied into binding variables in the list **[a b]**. We could have used the following code instead but it is more verbose:

```
=> (for [x vv]
    (setv a (first x))
    (setv b (second x))
... (print a b))
0 T
1 h
2 e
3
4 b
 . . .
13 e
14 d
=>
```

Formatted Output

I suggest using the Python **format** method when you need to format output. In the following repl listing, you can see a few formatting options: insert any Hy data into a string (line 3), print values with a specific width and right justified (in line 5 the width for both values is 15 characters), print values with a specific width and left justified (in line 7), and limiting the number of characters values can be expressed as (in line 9 the object "cat" is expressed as just the first two characters and the value 3.14159 is expressed as just three numbers, the period not counting).

```
$ hy
hy 0.18.0 using CPython(default) 3.7.4 on Darwin
=> (.format "first: {} second: {}" "cat" 3.14159)
'first: cat second: 3.14159'
=> (.format "first: {:>15} second: {:>15}" "cat" 3.14159)
'first:             cat second:         3.14159'
=> (.format "first: {:15} second: {:15}" "cat" 3.14159)
'first: cat             second:         3.14159'
=> (.format "first: {:.2} second: {:.3}" "cat" 3.14159)
'first: ca second: 3.14'
=>
```

Notice the calling .**format** here returns a string value rather than writing to an output stream.

Importing Libraries from Different Directories on Your Laptop

I usually write applications by first implementing simpler low-level utility libraries that are often not in the same directory path as the application that I am working on. Let's look at a simple example of accessing the library **nlp_lib.hy** in the directory **hy-lisp-python/nlp** from the directory **hy-lisp-python/webscraping**:

```
1  Marks-MacBook:hy-lisp-python $ pwd
2  /Users/markw/GITHUB/hy-lisp-python
3  Marks-MacBook:hy-lisp-python $ cd webscraping
4  Marks-MacBook:webscraping $ hy
5  hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
6  => (import sys)
7  => (sys.path.insert 1 "../nlp")
8  => (import [nlp-lib [nlp]])
9  => (nlp "President George Bush went to Mexico and he had a very good meal")
10 {'text': 'President George Bush went to Mexico and he had a very good meal',
11   ...
12  'entities': [['George Bush', 'PERSON'], ['Mexico', 'GPE']]}
13 => (import [coref-nlp-lib [coref-nlp]])
14 => (coref-nlp "President George Bush went to Mexico and he had a very good meal")
15 {'corefs': 'President George Bush went to Mexico and President George Bush had a ver\
16 y good meal',  ...  }}}
17 =>
```

Here I did not install the library **nlp_lib.hy** using Python setuptools (which I don't cover in this book, you can read the documentation[7]) as a library on the system. I rely on relative paths between the library directory and the application code that uses the library.

On line 6 I am inserting the library directory into the Python system load path so the import statement on line 8 can find the **nlp-lib** library and on line 13 can find the **coref-nlp-lib** library.

Using Closures

Function definitions can capture values defined outside of a function and even change the captured value as seen in this example (file **closure_example.hy** in the directory **hy-lisp-python/misc**):

[7]https://setuptools.readthedocs.io

```
#!/usr/bin/env hy

(require [hy.contrib.walk [let]])

(let [x 1]
  (defn increment []
    (setv x (+ x 1))
    x))

(print (increment))
(print (increment))
(print (increment))
```

That produces:

```
2
3
4
```

Using closures is often a good alternative to object oriented programming for maintaining private state that only one or a few functions (that are defined inside the closure) are allowed to access and modify. In the last example the **let** statement could have defined more than one variable with initial values and many functions could have been defined to perform various calculations with the values of these captured variables and/or change the values of captured variables. This effectively hides the variables defined in the **let** statement from code outside of the let statement but the functions are accessible from outside the **let** statement.

Hy Looks Like Clojure: How Similar Are They?

Clojure[8] is a dynamic general purpose Lisp language for the JVM. One of the great Clojure features is support of immutable data (read only after creation) that makes multi-threaded code easier to write and maintain.

Unfortunately, Clojure's immutable data structures cannot be easily implemented efficiently in Python so the Hy language does not support immutable data, except for tuples. Otherwise the syntax for defining functions, using maps/hash tables/dictionaries, etc. is similar between the two languages.

The original Hy language developer Paul Tagliamonte was clearly inspired by Clojure.

The book **Serious Python** by Julien Danjou has an entire chapter (chapter 9) on the Python AST (abstract syntax tree), an introduction to Hy, and an interview with Paul Tagliamonte. Recommended!

[8]https://clojure.org/

This podcast[9] in 2015 interviews Hy developers Paul Tagliamonte, Tuukka Turto, and Morten Linderud. You can see the current Hy contributer list on github[10].

Plotting Data Using the Numpy and the Matplotlib Libraries

Data visualization is a common task when working with numeric data. In a later chapter on Deep Learning we will use two functions, the **relu** and **sigmoid** functions. Here we will use a few simple Hy language scripts to plot these functions.

The Numpy library supports what is called "broadcasting" in Python. In the function **sigmoid** that we define in the following REPL, we can pass either a single floating point number or a Numpy array as an argument. When we pass a Numpy array, then the function **sigmoid** is applied to each element of the Numpy array:

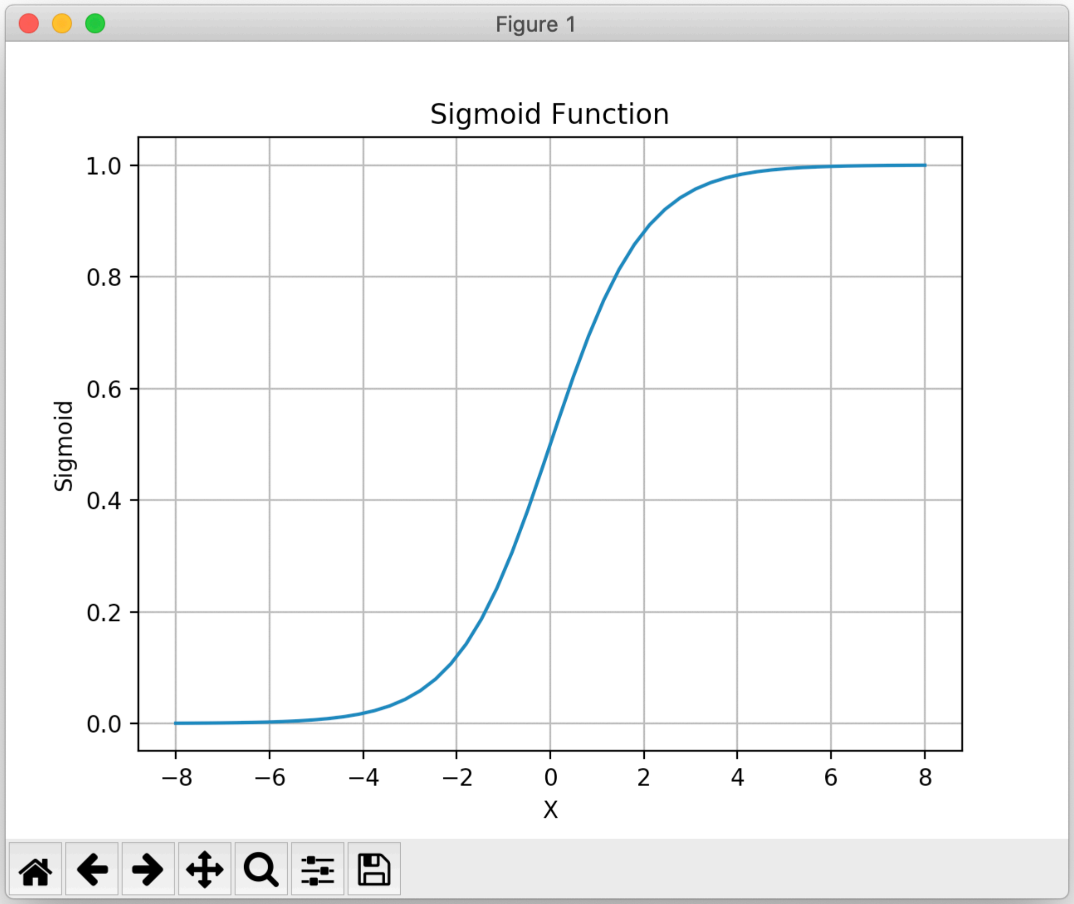
```
$ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import [numpy :as np])
=> (import [matplotlib.pyplot :as plt])
=>
=> (defn sigmoid [x]
...   (/ 1.0 (+ 1.0 (np.exp (- x)))))
=> (sigmoid 0.2)
0.549833997312478
=> (sigmoid 2)
0.8807970779778823
=> (np.array [-5 -2 0 2 5])
array([-5, -2,  0,  2,  5])
=> (sigmoid (np.array [-5 -2 0 2 5]))
array([0.00669285, 0.11920292, 0.5, 0.88079708, 0.99330715])
=>
```

The git repository directory **hy-lisp-python/matplotlib** contains two similar scripts for plotting the **sigmoid** and **relu** functions. Here is the script to plot the **sigmoid** function:

[9] https://www.pythonpodcast.com/episode-23-hylang-core-developers/
[10] https://github.com/hylang/hy/graphs/contributors

```
(import [numpy :as np])
(import [matplotlib.pyplot :as plt])

(defn sigmoid [x]
  (/ 1.0 (+ 1.0 (np.exp (- x)))))

(setv X (np.linspace -8 8 50))
(plt.plot X (sigmoid X))
(plt.title "Sigmoid Function")
(plt.ylabel "Sigmoid")
(plt.xlabel "X")
(plt.grid)
(plt.show)
```

The generated plot looks like this on macOS (Matplotlib is portable and also works on Windows and Linux):



Sigmoid Function

Bonus Points: Configuration for macOS and ITerm2 for Generating Plots Inline in a Hy REPL and Shell

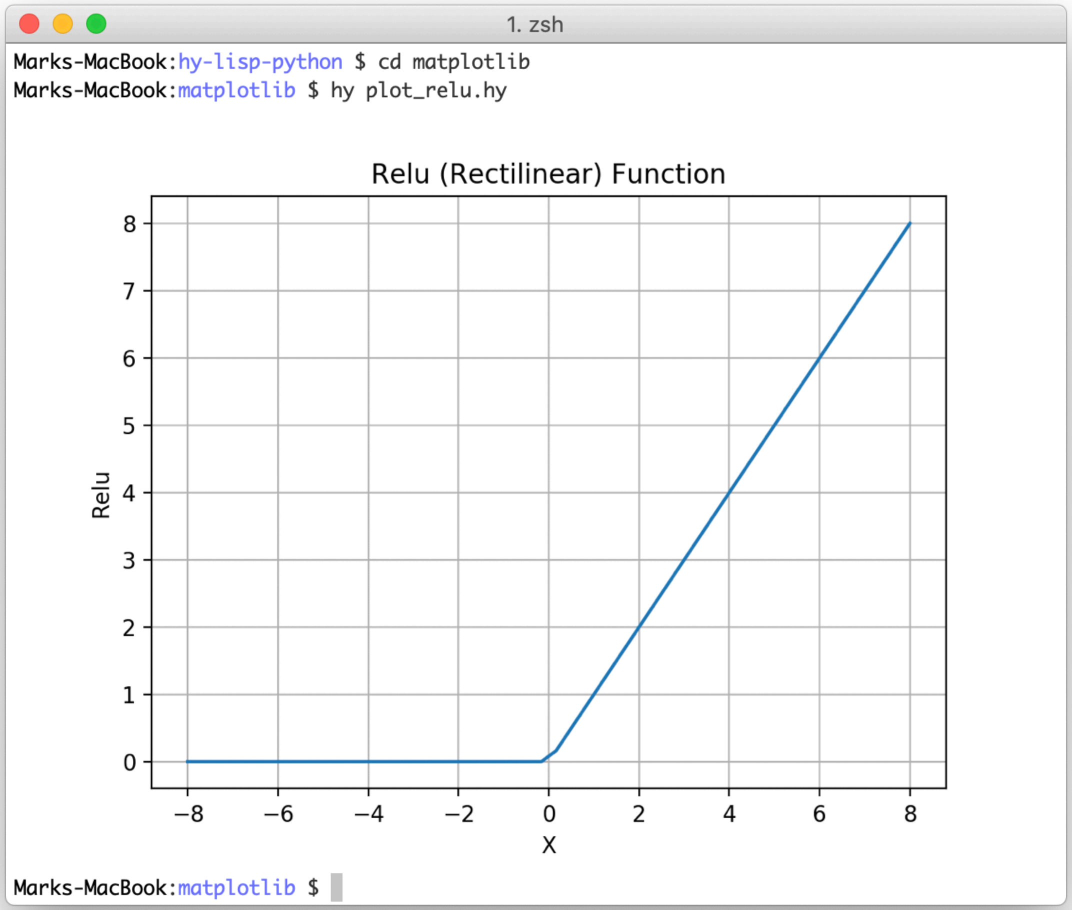
On the macOS ITerm2 terminal app and on most Linux terminal apps, it is possible to get inline matplotlib plots in a shell (bash, zsh, etc.), in Emacs, etc. This will take some setup work but it is well worth it especially if you work on remote servers via SSH or tmux. Here is the setup for macOS:

```
pip3 install itermplot
```

The add the following to your .profile, .bash_profile, or .zshrc (depending on your shell setup):

```
export MPLBACKEND="module://itermplot"
```

Here we run an example from the last section in a zsh shell (bash, etc. also should work):

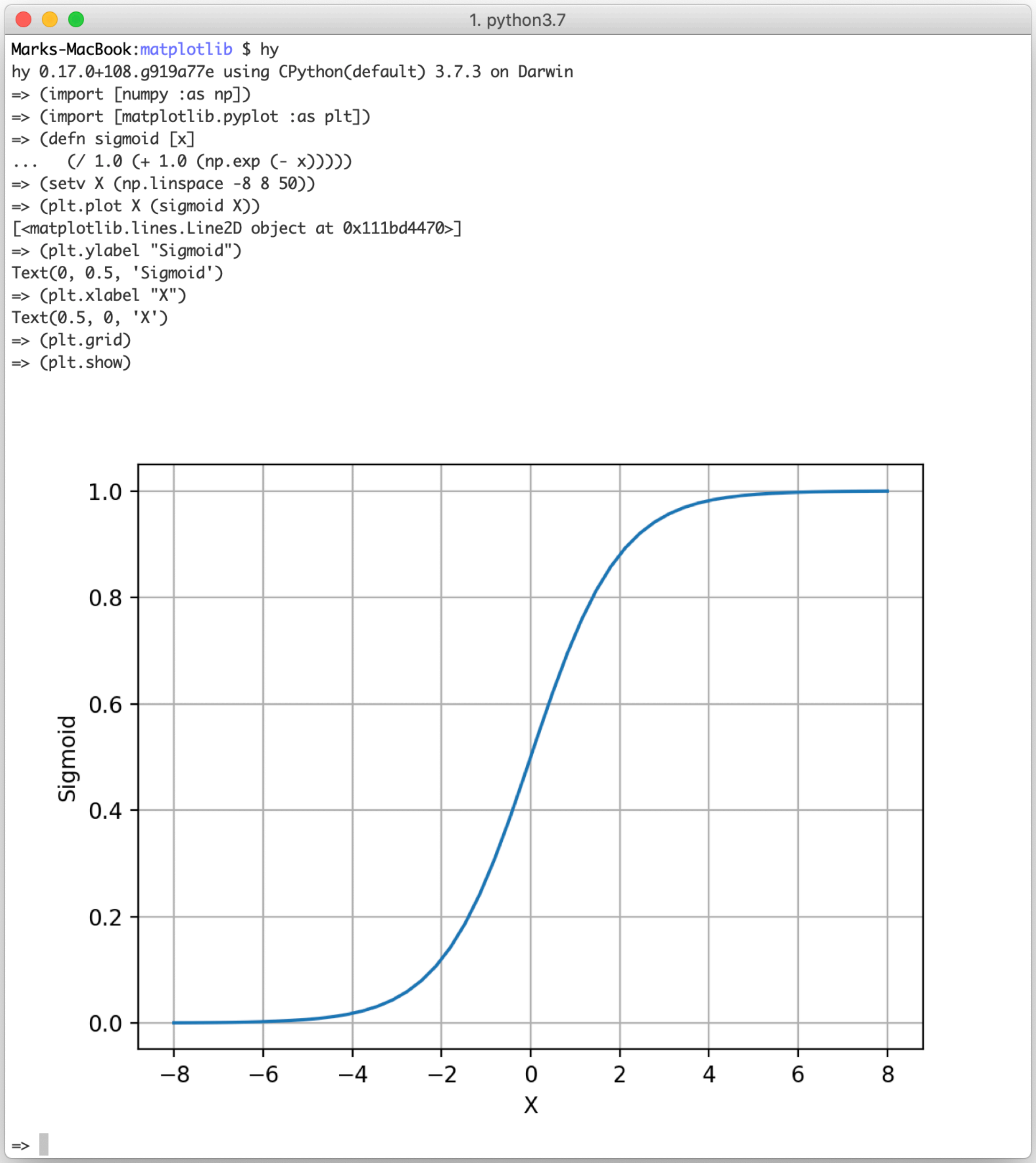


```
Marks-MacBook:hy-lisp-python $ cd matplotlib
Marks-MacBook:matplotlib $ hy plot_relu.hy
```

```
Marks-MacBook:matplotlib $
```

Inline matplotlib use in zsh shell in an ITerm on macOS

The best part of generating inline plots is during interactive REPL-based coding sessions:



```
Marks-MacBook:matplotlib $ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import [numpy :as np])
=> (import [matplotlib.pyplot :as plt])
=> (defn sigmoid [x]
...   (/ 1.0 (+ 1.0 (np.exp (- x)))))
=> (setv X (np.linspace -8 8 50))
=> (plt.plot X (sigmoid X))
[<matplotlib.lines.Line2D object at 0x111bd4470>]
=> (plt.ylabel "Sigmoid")
Text(0, 0.5, 'Sigmoid')
=> (plt.xlabel "X")
Text(0.5, 0, 'X')
=> (plt.grid)
=> (plt.show)
```

```
=>
```

Inline matplotlib use in a Hy REPL on macOS

If you use a Mac laptop to SSH into a remote Linux server you need to install **itermplot** and set the environment variable **MPLBACKEND** on the remote server.

Why Lisp?

Now that we have learned the basics of the Hy Lisp language in the last chapter, I would like to move our conversation to a broader question of why we would want to use Lisp. I want to start with my personal history of why I turned to Lisp languages in the late 1970s for almost all of my creative and research oriented development and later transitioned to also using Lisp languages in production.

I Hated the Waterfall Method in the 1970s but Learned to Love a Bottom-Up Programming Style

I graduated UCSB in the mid 1970s with a degree in Physics and took a job as a scientific programmer in the 100% employee owned company SAIC. My manager had a PhD in Computer Science and our team and the organization we were in used what is known as the waterfall method where systems were designed carefully from the top down, carefully planned mostly in their entirety, and then coded up. We, and the whole industry I would guess, wasted a lot of time with early planning and design work that had to be discarded or heavily modified after some experience implementing the system.

What would be better? I grew to love bottom-up programming. When I was given a new project I would start by writing and testing small procedures for low level operations, things I was sure I would need. I then aggregated the functionality into higher levels of control logic, access to data, etc. Finally I would write the high level application.

I mostly did this for a while writing code in FORTRAN at SAIC and using Algol for weekend consulting work Salk Institute, working on hooking up lab equipment to minicomputers in Roger Guillemin's lab (he won a Nobel Prize during that time, which was exciting). Learning Algol, a very different language than FORTRAN, helped broaden my perspectives.

I wanted a better programming language! I also wanted a more productive way to do my job both as a programmer and to make the best use of the few free hours a week that I had for my own research and learning about artificial intelligence (AI). I found my "better way" of development by adopting a bottom-up style that involves first writing low level libraries and utilities and then layering complete programs on top of well tested low level code.

First Introduction to Lisp

In the late 1970s I discovered a Lisp implementation on my company's DECsystem-10 timesharing computer. I had heard of Lisp in reading Bertram Raphael's book "THE THINKING COMPUTER.

Mind Inside Matter" and I learned Lisp on my own time and then, during lunch hour, taught a one day a week class to anyone at work who wanted to learn Lisp. After a few months of Lisp experience I received permission to teach an informal lunch time class to teach anyone working in my building who wanted to to learn Lisp on our DECsystem-10.

Lisp is the perfect language to support the type of bottom-up iterative programming style that I like.

Commercial Product Development and Deployment Using Lisp

My company, SAIC, identified AI as an important technology in the early 1980s. Two friends at work (Bob Beyster who founded SAIC and Joe Walkush who was our corporate treasurer and who liked Lisp from his engineering studies at MIT) arranged for the company to buy a hardware Lisp Machine, a Xerox 1108 for me. I ported Charles Forgy's expert system development language OPS5 to run on InterLisp-D on the Xerox Lisp Machines and we successfully sold this as a product. When Coral Common Lisp was released for the Apple Macintosh in 1984, I switched my research and development to the Mac and released ExperOPS5, which also sold well, and used Common Lisp to write the first prototypes for SAIC's ANSim neural network library. I converted my code to C++ to productize it. We also continued to use Lisp for IR&D projects and while working on the DARPA NMRD project.

Even though I proceeded to use C++ for much of my development, as well as writing C++ books for McGraw-Hill and J. Riley publishers, Lisp remained my "thinking and research" language.

Hy Macros Let You Extend the Hy Language in Your Programs

In my work I seldom use macros since I mostly write application type programs. Macros are useful for extending the syntax allowed for programs written in Lisp languages.

My most common use of macros is flexibly handling arguments without evaluating them. In the following example I want to write a macro **all-to-string** that takes a list of objects that can include undefined symbols. For example, if the variable **x** is undefined, then trying to evaluate **(print x 1)** will throw an error like:

```
NameError: name 'x' is not defined
```

The following listing shows my experiments in a Hy REPL to write the macro **all-to-string**:

```
 1  $ hy
 2  hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
 3  => (list (map str ["a" 4]))
 4  ['a', '4']
 5  => (.join " " (list (map str ["a" 4])))
 6  'a 4'
 7  => (defmacro foo2 [&rest x] x)
 8  <function foo2 at 0x10b91b488>
 9  => (foo2 1 2 3)
10  [1, 2, 3]
11  => (foo2 1 runpuppyrun 3)
12  Traceback (most recent call last):
13    File "stdin-3241d1d4f129e0da87f331bfe8f9f7aba903073a", line 1, in <module>
14      (foo2 1 runpuppyrun 3)
15  NameError: name 'runpuppyrun' is not defined
16  => (defmacro all-to-string [&rest x] (.join " " (list (map str x))))
17  <function all-to-string at 0x10b91b158>
18  => (all-to-string cater123 22)
19  'cater123 22'
20  => (all-to-string the boy ran to get 1 new helmet)
21  'the boy ran to get 1 new helmet'
22  => (all-to-string the boy "ran" to get 1 "new" helmet)
23  'the boy ran to get 1 new helmet'
24  =>
```

My first try in line 7 did not work, the macro just returning a function that echos the arguments but throws an error (line 50) when one of the arguments is a symbol with no definition. The second try on line 16 works as intended because we are mapping the function **str** (which coerces any argument into a string) over the argument list.

Performing Bottom Up Development Inside a REPL is a Lifestyle Choice

It is my personal choice to prefer a bottom up style of coding, effectively extending the Hy (or other Lisp) language to look like something that looks custom designed and built to solve a specific problem. This is possible in Lisp languages because once a function or macro is defined, it is for our purposes part of the Hy language. If, for example, you are writing a web application that uses a database then I believe that it makes sense to first write low level functions to perform operations that you know you will need, for example, for creating and updating customer data from the database, utility functions used in a web application (which we cover in the next chapter), etc. For the rest of your application, you use these new low level functions as if they were built into the language.

When I need to write a new low-level function, I start in a REPL and define variables (with test values) for what the function arguments will be. I then write the code for the function one line at a time using these “arguments” in expressions that will later be copied to a Hy source file. Immediately seeing results in a REPL helps me catch mistakes early, often a misunderstanding of the type or values of intermediate calculations. This style of coding works for me and I hope you like it also.

Writing Web Applications

Python has good libraries and frameworks for building web applications and here we will use the **Flask** library and framework "under the hood" and write two simple Hy Language web applications. We will start with a simple "Hello World" example in Python, see how to reformulate it in Hy, and then proceed with more complex examples that will show how to use HTML generating templates, sessions, and cookies to store user data for the next time they visit your web site. In a later chapter we will cover use of the SQLite and PostgreSQL databases which are commonly used to persist data for users in web applications. This pattern involves letting a user login and store a unique token for the user in a web browser cookie. In principle, you can do the same with web browser cookies but if a user visits your web site with a different browser or device then they will not have access to the data stored in cookies on a previous visit.

I like lightweight web frameworks. In Ruby I use Sinatra, in Haskell I use Spock, and when I built Java web apps I liked lightweight tools like JSP. Flask is simple but capable and using it from Hy is productive and fun. In addition to using lightweight frameworks I like to deploy web apps in the simplest way possible. We will close this chapter by discussing how to use the Heroku and Google Cloud Platform AppEngine platforms.

Getting Started With Flask: Using Python Decorators in Hy

You will need to install Flask using:

```
pip install flask
```

We will use the Hy macro **with-decorator** to replace Python code with annotations. Here the decorator **@app.route** is used to map a URI pattern with a Python callback function. In the following case we define the behavior when the index page of a web app is accessed:

```python
from flask import Flask

@app.route('/')
  def index():
     return "Hello World !")

app.run()
```

I first used Flask with the Hy language after seeing a post of code from HN user “volent”, seen in the file **flask_test.hy** in the directory **hy-lisp-python/webapp** that is functionally equivalent to the above Python code snippet:

```hy
#!/usr/bin/env hy

;; snippet by HN user volent:

(import [flask [Flask]])

(setv app (Flask "Flask test"))
(with-decorator (app.route "/")
  (defn index []
    "Hello World !"))
(app.run)
```

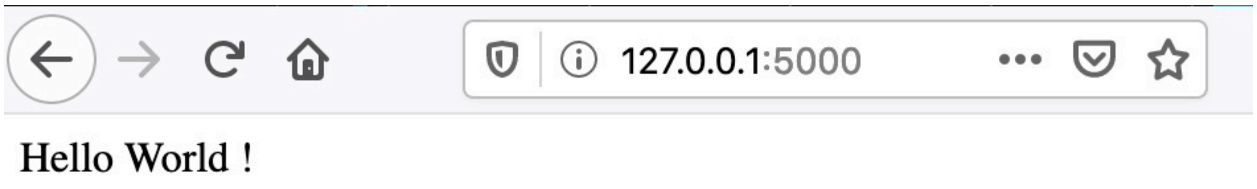
The Hy macro **with-decorator** macro is used to use Python style decorators in Hy applications.

I liked this example and after experimenting with the code, I then started using Hy and Flask. Please try running this example to make sure you are setup properly with Flask:

```
(base) Marks-MacBook:webapp $ ./flask_test.hy
 * Serving Flask app "Flask test" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Open http://127.0.0.1:5000/[11] in your web browser:

[11] http://127.0.0.1:5000/



Hello World !

Hello world Flask web app

Using Jinja2 Templates To Generate HTML

Jinja2[12] is a templating system that allows HTML markup to be supplemented with Python variable references and simple Python loops, etc. The values of application variables can be stored in a context and the HTML template has the variables values substituted with current values before returning a HTML response to the user's web browser.

By default Jinja2 templates are stored in a subdirectory named **templates**. The template for this example can be found in the file **hy-lisp-python/webapp/templates/template1.j2** that is shown here:

```
<html>
  <head>
    <title>Testing Jinja2 and Flask with the Hy language</title>
  </head>
  <body>
     {% if name %}
       <h1>Hello {{name}}</h1>
     {% else %}
       <h1>Hey, please enter your name!</h1>
     {% endif %}

    <form method="POST" action="/response">
      Name: <input type="text" name="name" required>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

[12]https://pypi.org/project/Jinja2/

Note that in line 6 we are using a Python **if** expression to check if the variable **name** is defined in the current app execution context.

In the context of a running Flask app, the following will render the above template with the variable **name** defined as **None**:

```
(render_template "template1.j2")
```

We can set values as named parameters for variables used in the template, for example:

```
(render_template "template1.j2" :name "Mark")
```

I am assuming that you understand the basics or HTML and also GET and POST operations in HTTP requests.

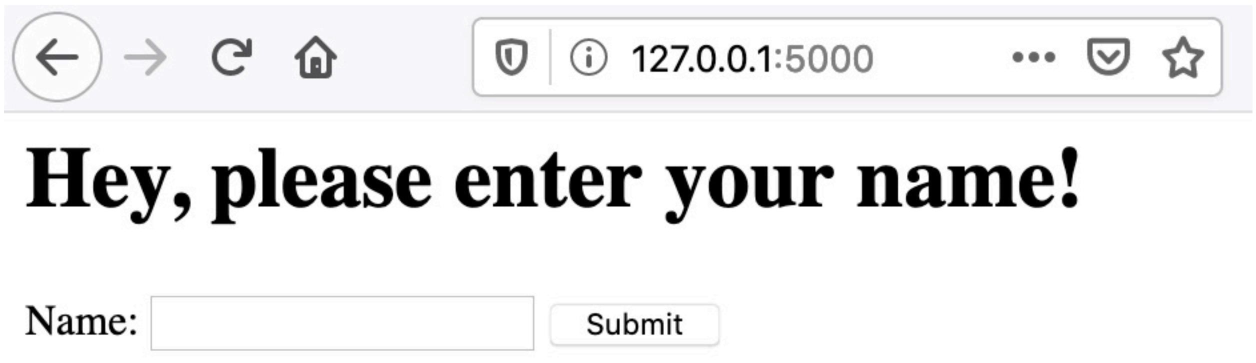
The following Flask web app defines behavior for rendering the template without the variable **name** set and also a HTML POST handler to pass the name entered on the HTML form back to the POST response handler:

```
#!/usr/bin/env hy

(import [flask [Flask render_template request]])

(setv app (Flask "Flask and Jinja2 test"))

(with-decorator (app.route "/")
  (defn index []
    (render_template "template1.j2")))

(with-decorator (app.route "/response" :methods ["POST"])
  (defn response []
    (setv name (request.form.get "name"))
    (print name)
    (render_template "template1.j2" :name name)))

(app.run)
```

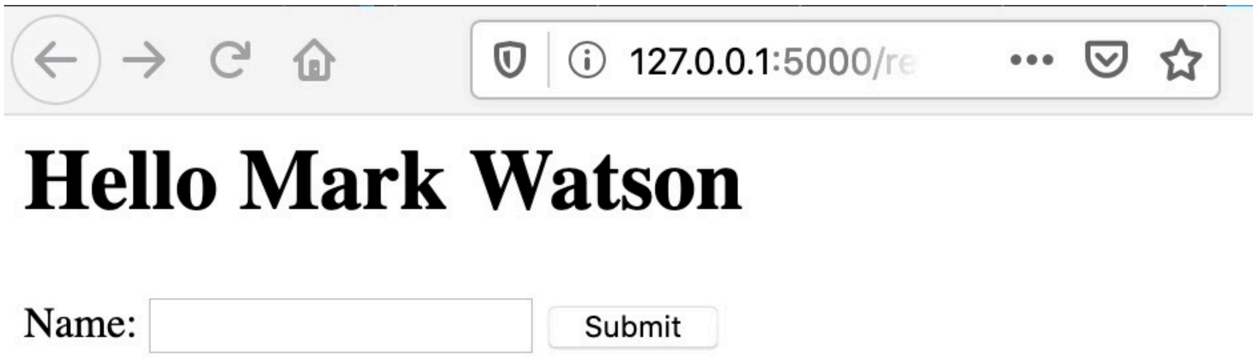
Please note that there is nothing special about the names inside the **with-decorator** code blocks: the functions **index** and **response** could have arbitrary names like **a123** an **b17**. I used the function names **index** and **response** because they help describe what the functions do.

Open http://127.0.0.1:5000/[13] in your web browser:

[13] http://127.0.0.1:5000/



Flask web app using a Jinja2 Template



Flask web app using a Jinja2 Template after entering my name and submitting the HTML input form

Handling HTTP Sessions and Cookies

There is a special variable **session** that Flask maintains for each client of a Flask web app. Different people using a web app will have independent sessions. In a web app, we can set a session value by treating the session for a given user as a dictionary:

```
=> (setv (get session "name") "Mark")
=> session
{'name': 'Mark'}
```

Inside a Jinja2 template you can use a simple Python expression to place a session variable's value into the HTML generated from a template:

```
{{ session['name'] }}
```

In a web app you can access the session using:

```
(get session "name")
```

In order to set the value of a named cookie, we can:

```
(import [flask [Flask render_template request make_response]])

(with-decorator (app.route "/response" :methods ["POST"])
  (defn response []
    (setv name (request.form.get "name"))
    (setv resp (make_reponse (render_template "template1.j2" :name name)))
    (resp.set_cookie "name" name)
    resp))
```

Values of named cookies can be retrieved using:

```
(request.cookies.get "name")
```

inside of a **with-decorator** form. The value for **request** is defined in the execution context by Flask when handling HTTP requests. Here is a complete example of handling cookies in the file *cookie_-test.hy*:

```
#!/usr/bin/env hy

(import [flask [Flask render_template request make-response]])

(setv app (Flask "Flask and Jinja2 test"))

(with-decorator (app.route "/")
  (defn index []
    (setv cookie-data (request.cookies.get "hy-cookie"))
    (print "cookie-data:" cookie-data)
    (setv a-response (render_template "template1.j2" :name cookie-data))
    a-response))

(with-decorator (app.route "/response" :methods ["POST"])
  (defn response []
    (setv name (request.form.get "name"))
    (print name)
    (setv a-response (make-response (render-template "template1.j2" :name name)))
    (a-response.set-cookie "hy-cookie" name)
    a-response))

(app.run)
```

I suggest that you not only try running this example as-is but also try changing the template, and generally experiment with the code. Making even simple code changes helps to understand the code better.

Deploying Hy Language Flask Apps to Google Cloud Platform AppEngine

The example for this section is in a separate github repository[14] that you should clone or copy to a new project for a starter project if you intend to deploy to AppEngine.

This AppEngine example is very similar to that in the last section except that it also serves a static asset and has a small Python stub main program to load the Hy language library and import the Hy language code.

Here is the Python stub main program:

```
import hy
import flask_test
from flask_test import app

if __name__ == '__main__':
    # Used when running locally only. When deploying to Google App
    # Engine, a webserver process such as Gunicorn will serve the app.
    app.run(host='localhost', port=9090, debug=True)
```

The Hy app is slightly different than we saw in the last section. On line 6 we specify the location of static assets and we do not call the **run()** method on the **app** object.

```
 1  (import [flask [Flask render_template request]])
 2  (import os)
 3  
 4  (setv port (int (os.environ.get "PORT" 5000)))
 5  
 6  (setv app (Flask "Flask test" :static_folder "./static" :static_url_path "/static"))
 7  
 8  (with-decorator (app.route "/")
 9    (defn index []
10      (render_template "template1.j2")))
11  
12  (with-decorator (app.route "/response" :methods ["POST"])
13    (defn response []
```

[14] https://github.com/mark-watson/hy-lisp-gcp-starter-project

```
    (setv name (request.form.get "name"))
    (render_template "template1.j2" :name name)))
```

I assume that you have some experience with GCP and have the following:

- GCP command line tools installed.
- You have created a new project on the GCP AppEngine console named something like hy-gcp-test (if you choose a name already in use, you wil get a warning).

After cloning or otherwise copying this project, you use the command line tools to deploy and test your Flask app:

```
gcloud auth login
gcloud config set project hy-gcp-test
gcloud app deploy
gcloud app browse
```

If you have problems, look at your logs:

```
gcloud app logs tail -s default
```

You can edit changes locally and test locally using:

```
python main.py
```

Any changes can be tested by deploying again:

```
gcloud app deploy
```

Please note that everytime you deploy, a new instance is created. You will want to use the GCP AppEngine console to remove old instances, and remove all instances when you are done.

Going forward

You can make a copy of this example, create a github repo, and follow the above directions as a first step to creating Hy language application on AppEngine. The Google Cloud Platform has many services that you can use in your app (using the Python APIs, called from your Hy program), including:

- Storage and Databases.
- Big Data.
- Machine Learning.

Deploying Hy Language Flask Apps to the Heroku Platform

The example for this section is in a separate github repository[15] that you should clone or otherwise use as starter project if you intend to deploy to the Heroku platform.

We use a Python stub program **wsgi.python** to make our Flask app work with the WSGI interface that Heroku uses:

```
import hy
import flask_test
from flask_test import app
```

The Heroku platform will call the **run()** method on the imported **app** object because of the settings in the Heroku **Proc** file for this project:

```
web: gunicorn 'wsgi:app' --log-file -
```

Here we are stating to the Heroku platform that we want the production-friendly **gunicorn** server to call the **run()** method on the **app** object that is defined in the **wsgi** module (here the module name is the prefix name of the Python WSGI handler file).

The Hy Flask app has a few changes from earlier examples. All changes are in line 3:

```
 1  (import [flask [Flask render_template request]])
 2
 3  (setv app (Flask "Flask test" :static_folder "./static" :static_url_path "/"))
 4
 5  (with-decorator (app.route "/")
 6    (defn index []
 7      (render_template "template1.j2")))
 8
 9  (with-decorator (app.route "/response" :methods ["POST"])
10    (defn response []
11      (setv name (request.form.get "name"))
12      (print name)
13      (render_template "template1.j2" :name name)))
```

You need to install the Heroku command line tools:

https://devcenter.heroku.com/categories/command-line[16]

After checking out this repo, do the following from this directory:

[15]https://github.com/mark-watson/hy-lisp-heroku-starter-project
[16]https://devcenter.heroku.com/categories/command-line

```
heroku login
heroku create
git push heroku master
```

If you have your Heroku account setup these commands will deploy this example.

You can look at the Heroku log files for your application using:

```
heroku logs --tail
```

You can open this Hello World app in your default web browser using:

```
heroku open
```

By default, your Hello World app will run on the free Heroku mode. You should still remove it when you are done:

- login to: https://dashboard.heroku.com/apps
- click on your application name
- click on the Settings tab
- scroll to the bottom of the page and use the option to delete the app

Going forward

You can make a copy of this example, create a github repo, and follow the above directions.

To test your Heroku setup locally or for development, you can use:

```
heroku local
```

The Heroku platform has a wide variety of supported services, including many third party services like data services[17] and Heroku and third party addons[18].

Wrap-up

I like to be able to implement simple things simply, without a lot of ceremony. Once you work through these examples I hope you feel that you can generate Hy and Flask based web apps quickly and with very little code required.

To return to the theme of bottom-up programming, I find that starting with short low level utility functions and placing them in a separate file makes reuse simple and makes future similar projects

[17]https://www.heroku.com/managed-data-services
[18]https://elements.heroku.com/addons

even easier. For each language I work with, I collect snippets of useful code and short utilities kept in separate files. When writing code I start looking in my snippets directory for the language I am using to implement low level functionality even before doing a web search. When I work in Common Lisp I keep all low level code that I have written in small libraries contained a single Quicklisp source root directory and for Python and Hy I use Python's **setuptools** library to generate libraries that are installed globally on my laptop for easy reuse. It is worth some effort to organize your work for future reuse.

Responsible Web Scraping

I put the word “Responsible” in the chapter title to remind you that just because it is easy (as we will soon see) to pull data from web sites, it is important to respect the property rights of web site owners and abide by their terms and conditions for use. This Wikipedia article on Fair Use[19] provides a good overview of using copyright material.

The web scraping code we develop here uses the Python BeautifulSoup and URI libraries.

For my work and research, I have been most interested in using web scraping to collect text data for natural language processing but other common applications include writing AI news collection and summarization assistants, trying to predict stock prices based on comments in social media which is what we did at Webmind Corporation in 2000 and 2001, etc.

Using the Python BeautifulSoup Library in the Hy Language

There are many good libraries for parsing HTML text and extracting both structure (headings, what is in bold font, etc.) and embedded raw text. I particularly like the Python Beautiful Soup library and we will use it here.

In line 4 for the following listing of file **get_web_page.hy**, I am setting the default user agent to a descriptive string “HyLangBook” but for some web sites you might need to set this to appear as a Firefox or Chrome browser (iOS, Android, Windows, Linux, or macOS). The function **get-raw-data** gets the entire contents of a web site as a single string value.

```
1  (import [urllib.request [Request urlopen]])
2
3  (defn get-raw-data-from-web [aUri
4                               &optional [anAgent {"User-Agent" "HyLangBook/1.0"}]]
5    (setv req (Request aUri :headers anAgent))
6    (setv httpResponse (urlopen req))
7    (setv data (.read httpResponse))
8    data)
```

Let’s test this function in a REPL:

[19] https://en.wikipedia.org/wiki/Fair_use

```
 1  $ hy
 2  hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
 3  => (import [get-page-data [get-raw-data-from-web]])
 4  => (get-raw-data-from-web "http://knowledgebooks.com")
 5  b'<!DOCTYPE html><html><head><title>KnowledgeBooks.com - research on the Knowledge M\
 6  anagement, and the Semantic Web ...'
 7  =>
 8  => (import [get-page-data [get-page-html-elements]])
 9  => (get-page-html-elements "http://knowledgebooks.com")
10  {'title': [<title>KnowledgeBooks.com - research on the Knowledge Management, and the\
11   Semantic Web </title>],
12  'a': [<a class="brand" href="#">KnowledgeBooks.com  </a>,  ...
13  =>
```

This REPL session shows the the function **get-raw-data-from-web** defined in the previous listing returns a web page as a string. In line 9 we use a function **get-page-html-elements** to find all elements in a string containing HTML. This function is defined in the next listing and shows how to parse and process the string contents of a web pages. Note: you will need to install the **lxml** library for this example (using pip or pip3 depending on your Python configuration):

```
pip install lxml
```

The following listing of file **get_page_data.hy** uses the Beautiful Soup library to parse the string data for HTML text from a web site. The function **get-page-html-elements** returns names and associated data with each element in HTML represented as a string (the extra code on lines 20-24 is just debug example code):

```
 1  (import [get_web_page [get-raw-data-from-web]])
 2  
 3  (import [bs4 [BeautifulSoup]])
 4  
 5  (defn get-element-data [anElement]
 6    {"text" (.getText anElement)
 7     "name" (. anElement name)
 8     "class" (.get anElement "class")
 9     "href" (.get anElement "href")})
10  
11  (defn get-page-html-elements [aUri]
12    (setv raw-data (get-raw-data-from-web aUri))
13    (setv soup (BeautifulSoup raw-data "lxml"))
14    (setv title (.find_all soup "title"))
15    (setv a (.find_all soup "a"))
16    (setv h1 (.find_all soup "h1"))
17    (setv h2 (.find_all soup "h2"))
18    {"title" title "a" a "h1" h1 "h2" h2})
19
20  (setv elements (get-page-html-elements "http://markwatson.com"))
21
22  (print (get elements "a"))
23
24  (for [ta (get elements "a")] (print (get-element-data ta)))
```

The function **get-element-data** defined in lines 5-9 accepts as an argument an HTML element object (as defined in the Beautiful soup library) and extracts data, if available, for text, name, class, and href values. The function **get-page-html-elements** defied in lines 11-18 accepts as an argument a string containing a URI and returns a dictionary (or map, or hashtable) containing lists of all **a**, **h1**, **h2**, and **title** elements in the web page pointed to by the input URI. You can modify **get-page-html-elements** to add additional HTML element types, as needed.

Here is the output (with many lines removed for brevity):

```
{'text': 'Mark Watson artificial intelligence consultant and author',
 'name': 'a', 'class': ['navbar-brand'], 'href': '#'}
{'text': 'Home page', 'name': 'a', 'class': None, 'href': '/'}
{'text': 'My Blog', 'name': 'a', 'class': None,
 'href': 'https://mark-watson.blogspot.com'}
{'text': 'GitHub', 'name': 'a', 'class': None,
 'href': 'https://github.com/mark-watson'}
{'text': 'Twitter', 'name': 'a', 'class': None, 'href': 'https://twitter.com/mark_l_\
watson'}
{'text': 'WikiData', 'name': 'a', 'class': None, 'href': 'https://www.wikidata.org/w\
iki/Q18670263'}
```

Getting HTML Links from the DemocracyNow.org News Web Site

I financially support and rely on both NPR.org and DemocracyNow.org news as my main sources of news so I will use their news sites for examples here and in the next section. Web sites differ so much in format that it is often necessary to build highly customized web scrapers for individual web sites and to maintain the web scraping code as the format of the site changes in time.

Before working through this example and/or the example in the next section use the file **Makefile** to fetch data:

```
make data
```

This should copy the home pages for both web sites to the files:

- democracynow_home_page.html (used here)
- npr_home_page.html (used for the example in the next section)

The following listing shows **democracynow_front_page.hy**

```
 1  #!/usr/bin/env hy
 2  
 3  (import [get-web-page [get-web-page-from-disk]])
 4  (import [bs4 [BeautifulSoup]])
 5  
 6  ;; you need to run 'make data' to fetch sample HTML data for dev and testing
 7  
 8  (defn get-democracy-now-links []
 9    (setv test-html (get-web-page-from-disk "democracynow_home_page.html"))
10    (setv bs (BeautifulSoup test-html :features "lxml"))
11    (setv all-anchor-elements (.findAll bs "a"))
12    (lfor e all-anchor-elements
13          :if (> (len (.get-text e)) 0)
14          (, (.get e "href") (.get-text e))))
15  
16  (if (= __name__ "__main__")
17    (for [[uri text] (get-democracy-now-links)]
18      (print uri ":" text)))
```

This simply prints our URIs and text (separated with the string “:”) for each link on the home page. On line 13 we discard any anchor elements that do not contain text. On line 14 the comma character at the start of the return list indicates that we are constructing a tuple. Lines 16-18 define a main function that is used when running this file art the command line. This is similar to how main functions can be defined in Python to allow a library file to also be run as a command line tool.

A few lines of output from today’s front page is:

```
/2020/1/7/the_great_hack_cambridge_analytica : Meet Brittany Kaiser, Cambridge Analy\
tica Whistleblower Releasing Troves of New Files from Data Firm
/2019/11/8/remembering_orangeburg_massacre_1968_south_carolina : Remembering the 196\
8 Orangeburg Massacre When Police Shot Dead Three Unarmed Black Students
/2020/1/15/democratic_debate_higher_education_universal_programs : Democrats Debate \
Wealth Tax, Free Public College & Student Debt Relief as Part of New Economic Plan
/2020/1/14/dahlia_lithwick_impeachment : GOP Debate on Impeachment Witnesses Intensi\
fies as Pelosi Prepares to Send Articles to Senate
/2020/1/14/oakland_california_moms_4_housing : Moms 4 Housing: Meet the Oakland Moth\
ers Facing Eviction After Two Months Occupying Vacant House
/2020/1/14/luis_garden_acosta_martin_espada : “Morir Soñando”: Martín Espada Reads P\
oem About Luis Garden Acosta, Young Lord & Community Activist
```

The URIs are relative to the root URI https://www.democracynow.org/.

Getting Summaries of Front Page from the NPR.org News Web Site

This example is similar to the example in the last section except that text from home page links is formatted to provide a daily news summary. I am assuming that you ran the example in the last section so the web site home pages have been copied to local files.

The following listing shows **npr_front_page_summary.hy**

```
#!/usr/bin/env hy

(import [get-web-page [get-web-page-from-disk]])
(import [bs4 [BeautifulSoup]])

;; you need to run 'make data' to fetch sample HTML data for dev and testing

(defn get-npr-links []
  (setv test-html (get-web-page-from-disk "npr_home_page.html"))
  (setv bs (BeautifulSoup test-html :features "lxml"))
  (setv all-anchor-elements (.findAll bs "a"))
  (setv filtered-a
    (lfor e all-anchor-elements
          :if (> (len (.get-text e)) 0)
          (, (.get e "href") (.get-text e))))
  filtered-a)

(defn create-npr-summary []
19    (setv links (get-npr-links))
20    (setv filtered-links (lfor [uri text] links :if (> (len (.strip text)) 40) (.strip\
21   text)))
22    (.join "\n\n" filtered-links))
23
24  (if (= __name__ "__main__")
25    (print (create-npr-summary)))
```

In lines 12-15 we are filtering out (or removing) all anchor HTML elements that do not contain text. The following shows a few lines of the generated output for data collected today:

```
January 16, 2020  Birds change the shape of their wings far more than
planes. The complexities of bird flight have posed a major design challenge
for scientists trying to translate the way birds fly into robots.

FBI Vows To Warn More Election Officials If Discovering A Cyberattack

January 16, 2020  The bureau was faulted after the Russian attack on the
2016 election for keeping too much information from state and local
authorities. It says it'll use a new policy going forward.

Ukraine Is Investigating Whether U.S. Ambassador Yovanovitch Was Surveilled

January 16, 2020  Ukraine's Internal Affairs Ministry says it's asking the
FBI to help determine whether international laws were broken, or "whether it
is just a bravado and a fake information" from a U.S. politician.

Electric Burn: Those Who Bet Against Elon Musk And Tesla Are Paying A Big Price

January 16, 2020  For years, Elon Musk skeptics have shorted Tesla stock, confident \
the electric carmaker was on the brink of disaster. Instead, share value has skyrock\
eted, costing short sellers billions.

TSA Says It Seized A Record Number Of Firearms At U.S. Airports Last Year
```

The examples seen here are simple but should be sufficient to get you started gathering text data from the web.

Using the Microsoft Bing Search APIs

You will need to register with Microsoft's Azure search service to use the material in this chapter. It is likely that you view search as a manual human-centered activity. I hope to expand your thinking to considering applications that automate search, finding information on the web, and automatically organizing information.

Getting an Access Key for Microsoft Bing Search APIs

You will need an Azure account. I use the Bing search APIs fairly often for research but I have never spent more than about a dollar a month and usually I get no bill at all. For personal use it is an inexpensive service.

Get started by going to the web page https://azure.microsoft.com/en-us/try/cognitive-services/[20] and sign up for an access key. The Search APIs signup is currently in the fourth tab in this web form. When you navigate to the Search APIs tab, select the option Bing Search APIs v7. You will get an API key that you need to store in an environment variable that you will soon need:

```
export BING_SEARCH_V7_SUBSCRIPTION_KEY=4e97234341d9891191c772b7371ad5b1
```

That is not my real subscription key!

After adding this to your **.profile** file (or **.zshrc**, or **.bashrc**, or etc.), open a new terminal window and make sure the following works for you:

```
$ hy
hy 0.18.0 using CPython(default) 3.7.4 on Darwin
=> (import os)
=> (get os.environ "BING_SEARCH_V7_SUBSCRIPTION_KEY")
'4e97234341d9891191c772b7371ad5b1'
=>
```

Example Search Script

It takes very little Hy code to access the Bing search APIs. We will look at a long example script that expects a single command line argument that is a string containing search terms. The following example script shows you how to make a search query that requests search results in JSON format. We also look at parsing the returned JSON data. I formatted this listing to fit the page width:

[20] https://azure.microsoft.com/en-us/try/cognitive-services/

```
#!/usr/bin/env hy

(import json)
(import os)
(import sys)
(import [pprint [pprint]])
(import requests)

;; Add your Bing Search V7 subscription key and
;; the endpoint to your environment variables.
(setv subscription_key (get os.environ "BING_SEARCH_V7_SUBSCRIPTION_KEY"))
(setv endpoint "https://api.cognitive.microsoft.com/bing/v7.0/search")

;; Query term(s) to search for.
(setv query (get sys.argv 1)) ;; an example: "site:wikidata.org Sedona Arizona"

;; Construct a request
(setv mkt "en-US")
(setv params { "q" query "mkt" mkt })
(setv headers { "Ocp-Apim-Subscription-Key" subscription_key })

;; Call the API
(setv response (requests.get endpoint :headers headers :params params))

(print "\nFull JSON response from Bing search query:\n")
(pprint (response.json))

;; pull out resuts and print them individually:

(setv results (get (response.json) "webPages"))

(print "\nResults from the key 'webPages':\n")
(pprint results)

(print "\nDetailed printout from the first search result:\n")

(setv result-list (get results "value"))
(setv first-result (first result-list))

(print "\nFirst result, all data:\n")
(pprint first-result)

(print "\nSummary of first search result:\n")
```

```
(pprint (get first-result "displayUrl"))

(if (in "displayUrl" first-result)
    (print
     (.format
       " key: {:15} \t:\t {}" "displayUrl"
       (get first-result "displayUrl"))))
(if (in "language" first-result)
    (print
      (.format " key: {:15} \t:\t {}" "language"
      (get first-result "language"))))
(if (in "name" first-result)
    (print
      (.format
        " key: {:15} \t:\t {}" "name"
        (get first-result "name"))))
```

You can use search hints like "site:wikidata.org" to only search specific web sites. In the following example I use the search query:

```
"site:wikidata.org Sedona Arizona"
```

This example generates 364 lines of output so I only show a few selected lines here:

```
$ ./bing.hy "site:wikidata.org Sedona Arizona" | wc -l
     364
$ ./bing.hy "site:wikidata.org Sedona Arizona"

Full JSON response from Bing search query:

{'_type': 'SearchResponse',
 'queryContext': {'originalQuery': 'site:wikidata.org Sedona Arizona'},

   ...

Results from the key 'webPages':

{'totalEstimatedMatches': 27,
 'value': [{'about': [{'name': 'Sedona'}, {'name': 'Sedona'}],
            'dateLastCrawled': '2020-05-24T00:04:00.0000000Z',
            'displayUrl': 'https://www.wikidata.org/wiki/Q80041',
    ...
```

```
Summary of first search result:

'https://www.wikidata.org/wiki/Q80041'
 key: displayUrl                  :         https://www.wikidata.org/wiki/Q80041
 key: language                    :         en
 key: name                        :         Sedona - Wikidata
```

Wrap-up

In addition to using automated web scraping to get data for my personal research, I often use automated web search. I find the Microsoft's Azure Bing search APIs are the most convenient to use and I like paying for services that I use. The search engine Duck Duck Go also provides free search APIs but even though I use Duck Duck Go for 90% of my manual web searches, when I build automated systems I prefer to rely on services that I pay for.

Deep Learning

Most of my professional career since 2014 has involved Deep Learning, mostly with TensorFlow using the Keras APIs. In the late 1980s I was on a DARPA neural network technology advisory panel for a year, I wrote the first prototype of the SAIC ANSim neural network library commercial product, and I wrote the neural network prediction code for a bomb detector my company designed and built for the FAA for deployment in airports. More recently I have used GAN (generative adversarial networks) models for synthesizing numeric spreadsheet data and LSTM (long short term memory) models to synthesize highly structured text data like nested JSON and for NLP (natural language processing). I have 55 USA and several European patents using neural network and Deep Learning technology.

The Hy language utilities and example programs we develop here all use TensorFlow and Keras "under the hood" to do the heavy lifting. Keras is a simpler to use API for TensorFlow and I usually use Keras rather than the lower level TensorFlow APIs.

There are other libraries and frameworks that might interest you in addition to TensorFlow and Keras. I particularly like the Flux library for the Julia programming language. Currently Python has the most comprehensive libraries for Deep Learning but other languages that support differential computing (more on this later) like Julia and Swift may gain popularity in the future.

Here we will learn a vocabulary for discussing Deep Learning neural network models, look at possible architectures, and show two Hy language examples that should be sufficient to get you used to using Keras with the Hy language. If you already have Deep Learning application development experience you might want to skip the following review material and skip to the Hy language examples.

If you want to use Deep Learning professionally, there are two specific online resources that I recommend: Andrew Ng leads the efforts at deeplearning.ai[21] and Jeremy Howard leads the efforts at fast.ai[22]. Here I will show you how to use a few useful techniques. Andrew and Jeremy will teach you skills that may lead a professional level of expertise if you take their courses.

There are many Deep Learning neural architectures in current practical use; a few types that I use are:

- Multi-layer perceptron networks with many fully connected layers. An input layer contains placeholders for input data. Each element in the input layer is connected by a two-dimensional weight matrix to each element in the first hidden layer. We can use any number of fully connected hidden layers, with the last hidden layer connected to an output layer.

[21]https://www.deeplearning.ai/
[22]https://www.fast.ai/

- Convolutional networks for image processing and text classification. Convolutions, or filters, are small windows that can process input images (filters are two-dimensional) or sequences like text (filters are one-dimensional). Each filter uses a single set of learned weights independent of where the filter is applied in an input image or input sequence.
- Autoencoders have the same number of input layer and output layer elements with one or more hidden fully connected layers. Autoencoders are trained to produce the same output as training input values using a relatively small number of hidden layer elements. Autoencoders are capable of removing noise in input data.
- LSTM (long short term memory) process elements in a sequence in order and are capable of remembering patterns that they have seen earlier in the sequence.
- GAN (generative adversarial networks) models comprise two different and competing neural models, the generator and the discriminator. GANs are often trained on input images (although in my work I have applied GANs to two-dimensional numeric spreadsheet data). The generator model takes as input a "latent input vector" (this is just a vector of specific size with random values) and generates a random output image. The weights of the generator model are trained to produce random images that are similar to how training images look. The discriminator model is trained to recognize if an arbitrary output image is original training data or an image created by the generator model. The generator and discriminator models are trained together.

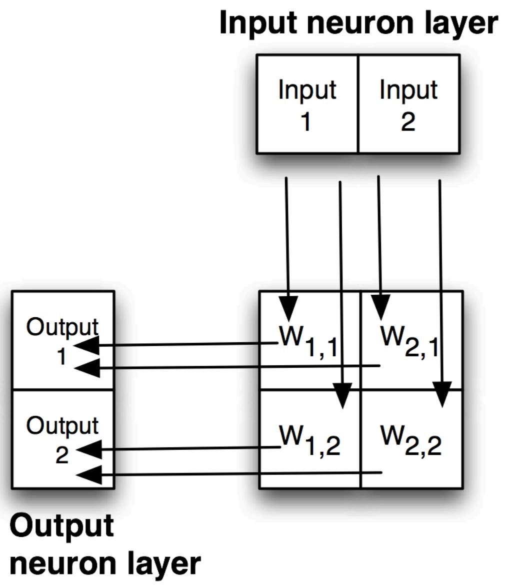
The core functionality of libraries like TensorFlow are written in C++ and take advantage of special hardware like GPUs, custom ASICs, and devices like Google's TPUs. Most people who work with Deep Learning models don't need to even be aware of the low level optimizations used to make training and using Deep Learning models more efficient. That said, in the following section I am going to show you how simple neural networks are trained and used.

Simple Multi-layer Perceptron Neural Networks

I use the terms Multi-layer perceptron neural networks, backpropagation neural networks and delta-rule networks interchangeably. Backpropagation refers to the model training process of calculating the output errors when training inputs are passed in the forward direction from input layer, to hidden layers, and then to the output layer. There will be an error which is the difference between the calculated outputs and the training outputs. This error can be used to adjust the weights from the last hidden layer to the output layer to reduce the error. The error is then backprogated backwards through the hidden layers, updating all weights in the model. I have detailed example code in any of my older artificial intelligence books. Here I am satisfied to give you an intuition to how simple neural networks are trained.

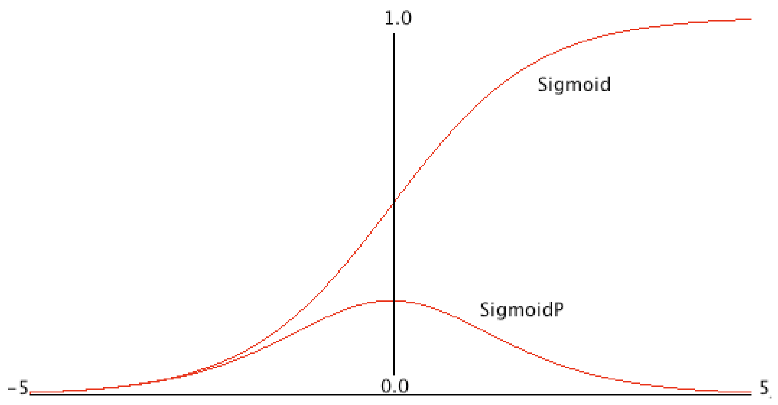
The basic idea is that we start with a network initialized with random weights and for each training case we propagate the inputs through the network towards the output neurons, calculate the output errors, and back-up the errors from the output neurons back towards the input neurons in order to make small changes to the weights to lower the error for the current training example. We repeat this process by cycling through the training examples many times.

The following figure shows a simple backpropagation network with one hidden layer. Neurons in adjacent layers are connected by floating point connection strength weights. These weights start out as small random values that change as the network is trained. Weights are represented in the following figure by arrows; in the code the weights connecting the input to the output neurons are represented as a two-dimensional array.



Example Backpropagation network with One Hidden Layer

Each non-input neuron has an activation value that is calculated from the activation values of connected neurons feeding into it, gated (adjusted) by the connection weights. For example, in the above figure, the value of Output 1 neuron is calculated by summing the activation of Input 1 times weight W1,1 and Input 2 activation times weight W2,1 and applying a “squashing function” like Sigmoid or Relu (see figures below) to this sum to get the final value for Output 1’s activation value. We want to flatten activation values to a relatively small range but still maintain relative values. To do this flattening we use the Sigmoid function that is seen in the next figure, along with the derivative of the Sigmoid function which we will use in the code for training a network by adjusting the weights.



Sigmoid Function and Derivative of Sigmoid Function (SigmoidP)

Simple neural network architectures with just one or two hidden layers are easy to train using backpropagation and I have from scratch code for this several of my previous books. You can see Java and Common Lisp from-scratch implementations in two of my books that you can read online: Practical Artificial Intelligence Programming With Java[23] and Loving Common Lisp, or the Savvy Programmer's Secret Weapon[24]. However, here we are using Hy to write models using the TensorFlow framework which has the huge advantage that small models you experiment with on your laptop can be scaled to more parameters (usually this means more neurons in hidden layers which increases the number of weights in a model) and run in the cloud using multiple GPUs.

Except for pendantic purposes, I now never write neural network code from scratch, instead I take advantage of the many person-years of engineering work put into the development of frameworks like TensorFlow, PyTorch, mxnet, etc. We now move on to two examples built with TensorFlow.

Deep Learning

Deep Learning models are generally understood to have many more hidden layers than simple multi-layer perceptron neural networks and often comprise multiple simple models combined together in series or in parallel. Complex architectures can be iteratively developed by manually adjusting the size of model components, changing the components, etc. Alternatively, model architecture search can be automated. At Capital One I used Google's AdaNet project[25] that efficiently searches for effective model architectures inside a single TensorFlow session. The model architecture used here is simple: one input layer representing the input values in a sample of University of Wisconsin cancer data, one hidden layer, and an output layer consisting of one neuron whose activation value will be interpreted as a prediction of benign or malignant.

The material in this chapter is intended to serve two purposes:

- If you are already familiar with Deep Learning and TensorFlow then the examples here will serve to show you how to call the TensorFlow APIs from Hy.

[23]https://leanpub.com/javaai
[24]https://leanpub.com/lovinglisp
[25]https://github.com/tensorflow/adanet

- If you have little or no exposure with Deep Learning then the short Hy language examples will provide you with concise code to experiment with and you can then decide to study further.

Once again, I recommend that you consider taking two online Deep Learning course sequences. For no cost, Jeremy Howard provides lessons at fast.ai[26] that are very good and the later classes use PyTorch which is a framework that is similar to TensorFlow. For a modest cost Andrew Ng provides classes at deeplearning.ai[27] that use TensorFlow. I have been working in the field of machine learning since the 1980s, but I still take Andrew's online classes to stay up-to-date. In the last eight years I have taken his Stanford University machine learning class twice and also his complete course sequence using TensorFlow. I have also worked through much of Jeremy's material. I recommend both course sequences without reservation.

Using Keras and TensorFlow to Model The Wisconsin Cancer Data Set

The University of Wisconsin cancer database has 646 samples. Each sample has 9 input values and one output value, the target output class (0 for benign, 1 for cancer):

- 0 Clump Thickness 1 - 10
- 1 Uniformity of Cell Size 1 - 10
- 2 Uniformity of Cell Shape 1 - 10
- 3 Marginal Adhesion 1 - 10
- 4 Single Epithelial Cell Size 1 - 10
- 5 Bare Nuclei 1 - 10
- 6 Bland Chromatin 1 - 10
- 7 Normal Nucleoli 1 - 10
- 8 Mitoses 1 - 10
- 9 Class (0 for benign, 1 for malignant)

We will use separate training and test files **hy-lisp-python/deeplearning/train.csv** and **hy-lisp-python/deeplearning/test.csv**. Here are a few samples from the training file:

[26]https://fast.ai
[27]https://www.deeplearning.ai/

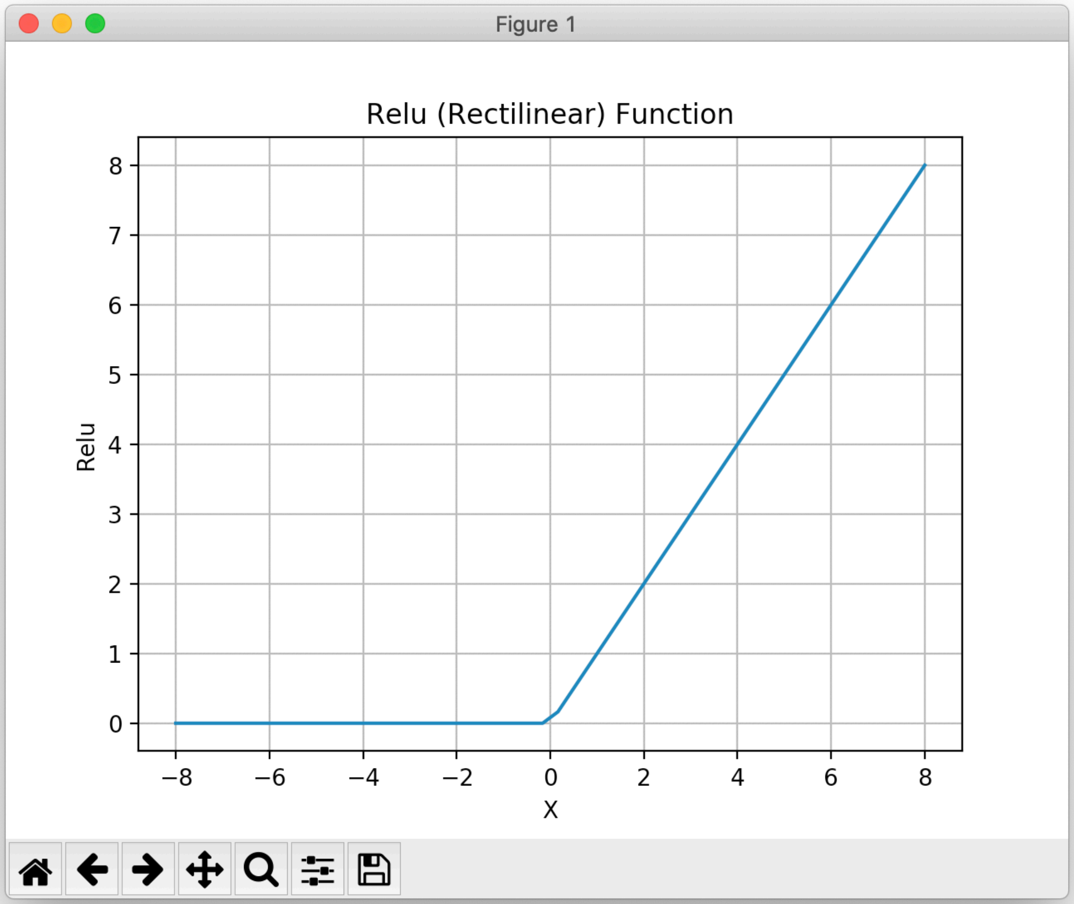
```
6,2,1,1,1,1,7,1,1,0
2,5,3,3,6,7,7,5,1,1
10,4,3,1,3,3,6,5,2,1
6,10,10,2,8,10,7,3,3,1
5,6,5,6,10,1,3,1,1,1
1,1,1,1,2,1,2,1,2,0
3,7,7,4,4,9,4,8,1,1
1,1,1,1,2,1,2,1,1,0
```

After you look at this data, if you did not have much experience with machine learning then it might not be obvious how to build a model to accept a sample for a patient like we see in the Wisconsin data set and then predict if the sample implies benign or cancerous outcome for the patient. Using TensorFlow with a simple neural network model, we will implement a model in about 40 lines of Hy code to implement this example.

Since there are nine input values we will need nine input neurons that will represent the input values for a sample in either training or separate test data. These nine input neurons (created in lines 9-10 in the following listing) will be completely connected to twelve neurons in a hidden layer. Here, completely connected means that each of the nine input neurons is connected via a weight to each hidden layer neuron. There are 9 * 12 = 108 weights between the input and hidden layers. There is a single output layer neuron that is connected to each hidden layer neuron.

Notice that in lines 12 and 14 in the following listing that we specify a **relu** activation function while the activation function connecting the hidden layer to the output layer uses the **sigmoid** activation function that we saw plotted earlier.

There is an example in the git example repo directory **hy-lisp-python/matplotlib** in the file **plot_-relu.hy** that generated the following figure:



Relu Function

The following listing shows the use of the Keras TensorFlow APIs to build a model (lines 9-19) with one input layer, two hidden layers, and an output layer with just one neuron. After we build the model, we define two utility functions **train** (lines 21-23) to train a model given training inputs (**x** argument**) and corresponding training outputs (**y** argument), and we also define **predict** (lines 25-26) using a trained model to make a cancer or benign prediction given test input values (**x-data** argument).

Lines 28-33 show a utility function **load-data** that loads a University of Wisconsin cancer data set CSV file, scales the input and output values to the range [0.0, 1.0] and returns a list containing input (**x-data**) and target output data (**y-data**). You may want to load this example in a REPL and evaluate **load-data** on one of the CSV files.

The function **main** (lines 35-45) loads training and test (evaluation of model accuracy on data not used for training), trains a model, and then tests the accuracy of the model on the test (evaluation) data:

```
#!/usr/bin/env hy

(import argparse os)
(import keras
        keras.utils.data-utils)

(import [pandas [read-csv]])

(defn build-model []
  (setv model (keras.models.Sequential))
  (.add model (keras.layers.core.Dense 9
               :activation "relu"))
  (.add model (keras.layers.core.Dense 12
               :activation "relu"))
  (.add model (keras.layers.core.Dense 1
               :activation "sigmoid"))
  (.compile model :loss      "binary_crossentropy"
                  :optimizer (keras.optimizers.RMSprop))
  model)

(defn train [batch-size model x y]
  (for [it (range 50)]
    (.fit model x y :batch-size batch-size :epochs 10 :verbose False)))

(defn predict [model x-data]
    (.predict model x-data))

(defn load-data [file-name]
  (setv all-data (read-csv file-name :header None))
  (setv x-data10 (. all-data.iloc [(, (slice 0 10) [0 1 2 3 4 5 6 7 8])] values))
  (setv x-data (* 0.1 x-data10))
  (setv y-data (. all-data.iloc [(, (slice 0 10) [9])] values))
  [x-data y-data])

(defn main []
  (setv xyd (load-data "train.csv"))
  (setv model (build-model))
  (setv xytest (load-data "test.csv"))
  (train 10 model (. xyd [0]) (. xyd [1]))
  (print "* predictions (calculated, expected):")
  (setv predictions (list (map first (predict model (. xytest [0])))))
  (setv expected (list (map first (. xytest [1]))))
  (print
    (list
      (zip predictions expected))))

(main)
```

The following listing shows the output:

```
$ hy wisconsin.hy
Using TensorFlow backend.
* predictions (calculated, expected):
[(0.9759052, 1), (0.99994254, 1), (0.8564741, 1), (0.95866203, 1), (0.03042546, 0), \
(0.21845636, 0), (0.99662805, 1), (0.08626339, 0), (0.045683343, 0), (0.9992156, 1)]
```

Let's look at the first test case: the "real" output from the training data is a value of 1 and the calculated predicted value (using the trained model) is 0.9759052. In making predictions, we can choose a cutoff value, 0.5 for example, and interpret any calculated prediction value less than the cutoff as a Boolean *false* prediction and calculated prediction value greater to or equal to the cutoff value is a Boolean *true* prediction.

Using a LSTM Recurrent Neural Network to Generate English Text Similar to the Philosopher Nietzsche's writing

We will translate a Python example program from Google's Keras documentation (listing of LSTM.py that is included with the example Hy code)[28] to Hy. This is a moderately long example and you can use the original Python and the translated Hy code as a general guide for converting other models implemented in Python using Keras that you want use in Hy. I have, in most cases, kept the same variable names to make it easier to compare the Python and Hy code.

Note that using the nietzsche.txt data set requires a fair amount of memory. If your computer has less than 16G of RAM, you might want to reduce the size of the training text by first running the following example until you see the printout "Create sentences and next_chars data…" then kill the program. The first time you run this program, the training data is fetched from the web and stored locally. You can manually edit the file ~/**.keras/datasets/nietzsche.txt** to remove 75% of the data by:

[28]https://keras.io/examples/lstm_text_generation/

```
    pushd ~/.keras/datasets/
    mv nietzsche.txt nietzsche_large.txt
    head -800 nietzsche_large.txt > nietzsche.txt
    popd
```

The next time you run the example, the Keras example data loading utilities will notice a local copy and even though the file now is much smaller, the data loading utilities will not download a new copy.

When I start training a new Deep Learning model I like to monitor system resources using the **top** command line activity, watching for page faults when training on a CPU which might indicate that I am trying to train too large of a model for my system memory. If you are using CUDA and a GPU then use the CUDA command line utilities for monitoring the state of the GPU utilization. It is beyond the scope of this introductory tutorial, but the tool TensorBoard[29] is very useful for monitoring the state of model training.

There are a few things that make the following example code more complex than the example using the University of Wisconsin cancer data set. We need to convert each character in the training data to a one-hot encoding which is a vector of all 0.0 values except for a single value of 1.0. I am going to show you a short REPL session so that you understand how this works and then we will look at the complete Hy code example.

```
$ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import [keras.callbacks [LambdaCallback]])
Using TensorFlow backend.
=> (import [keras.models [Sequential]])
=> (import [keras.layers [Dense LSTM]])
=> (import [keras.optimizers [RMSprop]])
=> (import [keras.utils.data_utils [get_file]])
=> (import [numpy :as np]) ;; note the syntax for aliasing a module name
=> (import random sys io)
=> (with [f (io.open "/Users/markw/.keras/datasets/nietzsche.txt" :encoding "utf-8")]
... (setv text (.read f)))
=> (cut text 98 130)
'philosophers, in so far as they '
=> (setv chars (sorted (list (set text))))
=> chars
['\n', ' ', '!', '"', "'", '(', ')', ',', '-', '.', '0', '1', '2', '3', '4', '5', '6\
', '7', '8', '9', ':', ';', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', '\
K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', '_', 'a', \
'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r',\
 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

[29]https://www.tensorflow.org/tensorboard/

```
22 => (setv char_indices (dict (lfor i (enumerate chars) (, (last i) (first i)))))
23 => char_indices
24 {'\n': 0, ' ': 1, '!': 2, '"': 3, "'": 4, '(': 5, ')': 6, ',': 7, '-': 8, '.': 9, '0\
25 ': 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': \
26 19, ':': 20, ';': 21, '?': 22, 'A': 23, 'B': 24, 'C': 25, 'D': 26, 'E': 27, 'F': 28,\
27  'G': 29, 'H': 30, 'I': 31, 'J': 32, 'K': 33, 'L': 34, 'M': 35, 'N': 36, 'O': 37, 'P\
28 ': 38, 'Q': 39, 'R': 40, 'S': 41, 'T': 42, 'U': 43, 'V': 44, 'W': 45, 'X': 46, 'Y': \
29 47, '_': 48, 'a': 49, 'b': 50, 'c': 51, 'd': 52, 'e': 53, 'f': 54, 'g': 55, 'h': 56,\
30  'i': 57, 'j': 58, 'k': 59, 'l': 60, 'm': 61, 'n': 62, 'o': 63, 'p': 64, 'q': 65, 'r\
31 ': 66, 's': 67, 't': 68, 'u': 69, 'v': 70, 'w': 71, 'x': 72, 'y': 73, 'z': 74}
32 => (setv indices_char (dict (lfor i (enumerate chars) i)))
33 => indices_char
34 {0: '\n', 1: ' ', 2: '!', 3: '"', 4: "'", 5: '(', 6: ')', 7: ',', 8: '-', 9: '.', 10\
35 : '0', 11: '1', 12: '2', 13: '3', 14: '4', 15: '5', 16: '6', 17: '7', 18: '8', 19: '\
36 9', 20: ':', 21: ';', 22: '?', 23: 'A', 24: 'B', 25: 'C', 26: 'D', 27: 'E', 28: 'F',\
37  29: 'G', 30: 'H', 31: 'I', 32: 'J', 33: 'K', 34: 'L', 35: 'M', 36: 'N', 37: 'O', 38\
38 : 'P', 39: 'Q', 40: 'R', 41: 'S', 42: 'T', 43: 'U', 44: 'V', 45: 'W', 46: 'X', 47: '\
39 Y', 48: '_', 49: 'a', 50: 'b', 51: 'c', 52: 'd', 53: 'e', 54: 'f', 55: 'g', 56: 'h',\
40  57: 'i', 58: 'j', 59: 'k', 60: 'l', 61: 'm', 62: 'n', 63: 'o', 64: 'p', 65: 'q', 66\
41 : 'r', 67: 's', 68: 't', 69: 'u', 70: 'v', 71: 'w', 72: 'x', 73: 'y', 74: 'z'}
42 => (setv maxlen 40)
43 => (setv s "Oh! I saw 1 dog (yesterday)")
44 => (setv x_pred (np.zeros [1 maxlen (len chars)]))
45 => (for [[t char] (lfor j (enumerate s) j)]
46 ... (setv (get x_pred 0 t (get char_indices char)) 1))
47 => x_pred
48 array([[[0., 0., 0., ..., 0., 0., 0.],
49         [0., 0., 0., ..., 0., 0., 0.],
50         [0., 0., 1., ..., 0., 0., 0.],   // here 1. is the third character "!"
51         ...,
52         [0., 0., 0., ..., 0., 0., 0.],
53         [0., 0., 0., ..., 0., 0., 0.],
54         [0., 0., 0., ..., 0., 0., 0.]]])
55 =>
```

For lines 48-54, each line represents a single character one-hot encoded. Notice how the third character shown on line 50 has a value of "1." at index 2, which corresponds to the one-hot encoding of the letter "!".

Now that you have a feeling for how one-hot encoding works, hopefully the following example will make sense to you. We will further discuss one-hot-encoding after the next code listing. For training, we take 40 characters (the value of the variable **maxlen**) at a time, and using one one-hot encode a character at a time as input and the target output will be the one-hot encoding of the following character in the input sequence. We are iterating on training the model for a while and then given a few characters of text, predict a likely next character - and keep repeating this process. The generated text is then used as input to the model to generate yet more text. You can repeat this process until you have generated sufficient text.

This is a powerful technique that I used to model JSON with complex deeply nested schemas and then generate synthetic JSON in the same schema as the training data. Here, training a model to mimic the philosopher Nietzsche's writing is much easier than learning highly structured data like JSON:

```
#!/usr/bin/env hy

;; This example was translated from the Python example in the Keras
;; documentation at: https://keras.io/examples/lstm_text_generation/
;; The original Python file LSTM.py is included in the directory
;; hy-lisp-python/deeplearning for reference.

(import [keras.callbacks [LambdaCallback]])
(import [keras.models [Sequential]])
(import [keras.layers [Dense LSTM]])
(import [keras.optimizers [RMSprop]])
(import [keras.utils.data_utils [get_file]])
(import [numpy :as np]) ;; note the syntax for aliasing a module name
(import random sys io)

(setv path
      (get_file        ;; this saves a local copy in ~/.keras/datasets
        "nietzsche.txt"
        :origin "https://s3.amazonaws.com/text-datasets/nietzsche.txt"))

(with [f (io.open path :encoding "utf-8")]
  (setv text (.read f))) ;; note: sometimes we use (.lower text) to
;;       convert text to all lower case
(print "corpus length:" (len text))

(setv chars (sorted (list (set text))))
(print "total chars (unique characters in input text):" (len chars))
(setv char_indices (dict (lfor i (enumerate chars) (, (last i) (first i)))))
(setv indices_char (dict (lfor i (enumerate chars) i)))

;; cut the text in semi-redundant sequences of maxlen characters
(setv maxlen 40)
(setv step 3) ;; when we sample text, slide sampling window 3 characters
```

```
(setv sentences (list))
(setv next_chars (list))

(print "Create sentences and next_chars data...")
(for [i (range 0 (- (len text) maxlen) step)]
  (.append sentences (cut text i (+ i maxlen)))
  (.append next_chars (get text (+ i maxlen))))

(print "Vectorization...")
(setv x (np.zeros [(len sentences) maxlen (len chars)] :dtype np.bool))
(setv y (np.zeros [(len sentences) (len chars)] :dtype np.bool))
(for [[i sentence] (lfor j (enumerate sentences) j)]
  (for [[t char] (lfor j (enumerate sentence) j)]
    (setv (get x i t (get char_indices char)) 1))
  (setv (get y i (get char_indices (get next_chars i))) 1))
(print "Done creating one-hot encoded training data.")

(print "Building model...")
(setv model (Sequential))
(.add model (LSTM 128 :input_shape [maxlen (len chars)]))
(.add model (Dense (len chars) :activation "softmax"))

(setv optimizer (RMSprop 0.01))
(.compile model :loss "categorical_crossentropy" :optimizer optimizer)

(defn sample [preds &optional [temperature 1.0]]
  (setv preds (.astype (np.array preds) "float64"))
  (setv preds (/ (np.log preds) temperature))
  (setv exp_preds (np.exp preds))
  (setv preds (/ exp_preds (np.sum exp_preds)))
  (setv probas (np.random.multinomial 1 preds 1))
  (np.argmax probas))

(defn on_epoch_end [epoch &optional not-used]
  (print)
  (print "----- Generating text after Epoch:" epoch)
  (setv start_index (random.randint 0 (- (len text) maxlen 1)))
  (for [diversity [0.2 0.5 1.0 1.2]]
    (print "----- diversity:" diversity)
    (setv generated "")
    (setv sentence (cut text start_index (+ start_index maxlen)))
    (setv generated (+ generated sentence))
    (print "----- Generating with seed:" sentence)
77      (sys.stdout.write generated)
78      (for [i (range 400)]
79        (setv x_pred (np.zeros [1 maxlen (len chars)]))
80        (for [[t char] (lfor j (enumerate sentence) j)]
81          (setv (get x_pred 0 t (get char_indices char)) 1))
82        (setv preds (first (model.predict x_pred :verbose 0)))
83        (setv next_index (sample preds diversity))
84        (setv next_char (get indices_char next_index))
85        (setv sentence (+ (cut sentence 1) next_char))
86        (sys.stdout.write next_char)
87        (sys.stdout.flush))
88      (print)))
89
90  (setv print_callback (LambdaCallback :on_epoch_end on_epoch_end))
91
92  (model.fit x y :batch_size 128 :epochs 60 :callbacks [print_callback])
```

In lines 52-54 we defined a model using the Keras APIs and in lines 56-57 compiled the model using a categorical crossentropy loss function with an RMSprop optimizer[30].

In lines 59-65 we define a function **sample** that takes a first required argument **preds** which is a one-hot predicted encoded character that might look like (maxlen or 40 values):

[2.80193929e-02 6.78635418e-01 7.85831537e-04 4.92034527e-03 . . . 6.62320468e-04 9.14627407e-03 2.31375365e-04]

Now, here the predicted one hot encoding values are not strictly 0 or 1, rather they are small floating point numbers of a single number much larger than the others. The largest number is 6.78635418e-01 at index 1 which corresponds to a one-hot encoding for a " " space character.

If we print out the number of characters in text and the unique list of characters (variable **chars**) in the training text file nietzsche.txt we see:

```
corpus length: 600893
['\n', ' ', '!', '"', "'", '(', ')', ',', '-', '.', '0', '1', '2', '3', '4', '5', '6\
', '7', '8', '9', ':', ';', '=', '?', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', '\
J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', \
'[', ']', '_', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',\
 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'Æ', 'ä', 'æ', 'é', 'ë'\
]
```

A review of one-hot encoding:

Let's review our earlier discussion of one-hot encoding with a simpler case. It is important to understand how we one-hot encode input text to inputs for the model and decode back one-hot

[30]https://keras.io/optimizers/

vectors to text when we use a trained model to generate text. It will help to see the dictionaries for converting characters to indices and then reverse indices to original characters as we saw earlier, some output removed:

```
char_indices:
 {'\n': 0, ' ': 1, '!': 2, '"': 3, "'": 4, '(': 5, ')': 6, ',': 7, '-': 8, '.': 9, '\
0': 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9':\
 19,
   . . .
 'f': 58, 'g': 59, 'h': 60, 'i': 61, 'j': 62, 'k': 63, 'l': 64, 'm': 65, 'n': 66, 'o\
': 67, 'p': 68, 'q': 69, 'r': 70, 's': 71, 't': 72, 'u': 73, 'v': 74, 'w': 75, 'x': \
76, 'y': 77, 'z': 78, 'Æ': 79, 'ä': 80, 'æ': 81, 'é': 82, 'ë': 83}
indices_char:
 {0: '\n', 1: ' ', 2: '!', 3: '"', 4: "'", 5: '(', 6: ')', 7: ',', 8: '-', 9: '.', 1\
0: '0', 11: '1', 12: '2', 13: '3', 14: '4', 15: '5', 16: '6', 17: '7', 18: '8', 19: \
'9',
   . . .
 'o', 68: 'p', 69: 'q', 70: 'r', 71: 's', 72: 't', 73: 'u', 74: 'v', 75: 'w', 76: 'x\
', 77: 'y', 78: 'z', 79: 'Æ', 80: 'ä', 81: 'æ', 82: 'é', 83: 'ë'}
```

We prepare the input and target output data in lines 43-48 in the last code listing. Using a short string, let's look in the next REPL session listing at how these input and output training examples are extracted for an input string:

```
Marks-MacBook:deeplearning $ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (setv text "0123456789abcdefg")
=> (setv maxlen 4)
=> (setv i 3)
=> (cut text i (+ i maxlen))
'3456'
=> (cut text (+ 1 maxlen))
'56789abcdefg'
=> (setv i 4)                  ;; i is the for loop variable for
=> (cut text i (+ i maxlen))  ;; defining sentences and next_chars
'4567'
=> (cut text (+ i maxlen))
'89abcdefg'
=>
```

So the input training sentences are each **maxlen** characters long and the **next-chars** target outputs each start with the character after the last character in the corresponding input training sentence.

This script pauses during each training epoc to generate text given diversity values of 0.2, 0.5, 1.0, and 1.2. The smaller the diversity value the more closely the generated text matches the training text. The generated text is more realistic after many training epocs. In the following, I list a highly edited copy of running through several training epochs. I only show generated text for diversity equal to 0.2:

```
----- Generating text after Epoch: 0
----- diversity: 0.2
----- Generating with seed: ocity. Equally so, gratitude.--Justice r
ocity. Equally so, gratitude.--Justice read in the become to the conscience the seen\
er and the conception that the becess of the power to the procentical that the becau\
se and the prostice of the prostice and the will to the conscience of the power of t\
he perhaps the self-distance of the all the soul and the world and the soul of the s\
oul of the world and the soul and an an and the profound the self-dister the all the\
 belief and the

----- Generating text after Epoch: 8
----- diversity: 0.2
----- Generating with seed: nations
laboring simultaneously under th
nations
laboring simultaneously under the subjection of the soul of the same to the subjecti\
on of the subjection of the same not a strong the soul of the spiritual to the same \
really the propers to the stree be the subjection of the spiritual that is to probab\
ly the stree concerning the spiritual the sublicities and the spiritual to the proce\
ssities the spirit to the soul of the subjection of the self-constitution and proper\
s to the

----- Generating text after Epoch: 14
----- diversity: 0.2
----- Generating with seed:  to which no other path could conduct us
 to which no other path could conduct us a stronger that is the self-delight and the\
 strange the soul of the world of the sense of the sense of the consider the such a \
state of the sense of the sense of the sense of such a sandine and interpretation of\
 the process of the sense of the sense of the sense of the soul of the process of th\
e world in the sense of the sense of the spirit and superstetion of the world the se\
nse of the

----- Generating text after Epoch: 17
----- diversity: 0.2
----- Generating with seed: hemselves although they could easily hav
hemselves although they could easily have been moral morality and the self-in which \
the self-in the world to the same man in the standard to the possibility that is to \
the strength of the sense-in the former the sense-in the special and the same man in\
 the consequently the soul of the superstition of the special in the end to the poss\
ible that it is will not be a sort of the superior of the superstition of the same m\
an to the same man
```

Here we trained on examples, translated to English, of the philosopher Nietzsche. I have used similar code to this example to train on highly structured JSON data and the resulting LSTM bsed model was usually able to generate similarly structured JSON. I have seen other examples where the training data was code in C++.

How is this example working? The model learns what combinations of characters tend to appear together and in what order.

In the next chapter we will use pre-trained Deep Learning models for natural language processing (NLP).

Natural Language Processing

I have been working in the field of Natural Language Processing (NLP) since 1985 so I 'lived through' the revolutionary change in NLP that has occurred since 2014: deep learning results out-classed results from previous symbolic methods.

I will not cover older symbolic methods of NLP here, rather I refer you to my previous books Practical Artificial Intelligence Programming With Java[31], [Loving Common Lisp, The Savvy Programmer's Secret Weapon[32], and Haskell Tutorial and Cookbook[33] for examples. We get better results using Deep Learning (DL) for NLP and the library **spaCy** (https://spacy.io[34]) that we use in this chapter provides near state of the art performance. The authors of **spaCy** frequently update it to use the latest breakthroughs in the field.

You will learn how to apply both DL and NLP by using the state-of-the-art full-feature library spaCy[35]. This chapter concentrates on how to use spaCy in the Hy language for solutions to a few selected problems in NLP that I use in my own work. I urge you to also review the "Guides" section of the spaCy documentation[36] where examples are in Python but after experimenting with the examples in this chapter you should have no difficulty in translating any spaCy Python examples to the Hy language.

If you have not already done so install the **spaCy** library and the full English language model:

```
pip install spacy
python -m spacy download en
```

You can use a smaller model (which requires loading "en_core_web_sm" instead of "en" in the following examples):

```
pip install spacy
python -m spacy download en_core_web_sm
```

Exploring the spaCy Library

We will use the Hy REPL to experiment with spaCy, Lisp style. The following REPL listings are all from the same session, split into separate listings so that I can talk you through the examples:

[31]https://leanpub.com/javaai
[32]https://leanpub.com/lovinglisp
[33]https://leanpub.com/haskell-cookbook
[34]https://spacy.io/
[35]https://spacy.io/
[36]https://spacy.io/usage

```
1  Marks-MacBook:nlp $ hy
2  hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
3  => (import spacy)
4  => (setv nlp-model (spacy.load "en"))
5  => (setv doc (nlp-model "President George Bush went to Mexico and he had a very good\
6   meal"))
7  => doc
8  President George Bush went to Mexico and he had a very good meal
9  => (dir doc)
10 ['_', '__bytes__', '__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__fo\
11 rmat__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init_\
12 _', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new\
13 __', '__pyx_vtable__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__\
14 setstate__', '__sizeof__', '__str__', '__subclasshook__', '__unicode__', '_bulk_merg\
15 e', '_py_tokens', '_realloc', '_vector', '_vector_norm', 'cats', 'char_span', 'count\
16 _by', 'doc', 'ents', 'extend_tensor', 'from_array', 'from_bytes', 'from_disk', 'get_\
17 extension', 'get_lca_matrix', 'has_extension', 'has_vector', 'is_nered', 'is_parsed'\
18 , 'is_sentenced', 'is_tagged', 'lang', 'lang_', 'mem', 'merge', 'noun_chunks', 'noun\
19 _chunks_iterator', 'print_tree', 'remove_extension', 'retokenize', 'sentiment', 'sen\
20 ts', 'set_extension', 'similarity', 'tensor', 'text', 'text_with_ws', 'to_array', 't\
21 o_bytes', 'to_disk', 'to_json', 'user_data', 'user_hooks', 'user_span_hooks', 'user_\
22 token_hooks', 'vector', 'vector_norm', 'vocab']
```

In lines 3-6 we import the spaCy library, load the English language model, and create a document from input text. What is a spaCy document? In line 9 we use the standard Python function **dir** to look at all names and functions defined for the object **doc** returned from applying a spaCy model to a string containing text. The value printed shows many built in "dunder" (double underscore attributes), and we can remove these:

In lines 23-26 we use the **dir** function again to see the attributes and methods for this class, but filter out any attributes containing the characters "__":

```
23 => (lfor
24 ... x (dir doc)
25 ... :if (not (.startswith x "__"))
26 ... x)
27 ['_', '_bulk_merge', '_py_tokens', '_realloc', '_vector', '_vector_norm', 'cats', 'c\
28 har_span', 'count_by', 'doc', 'ents', 'extend_tensor', 'from_array', 'from_bytes', '\
29 from_disk', 'get_extension', 'get_lca_matrix', 'has_extension', 'has_vector', 'is_ne\
30 red', 'is_parsed', 'is_sentenced', 'is_tagged', 'lang', 'lang_', 'mem', 'merge', 'no\
31 un_chunks', 'noun_chunks_iterator', 'print_tree', 'remove_extension', 'retokenize', \
32 'sentiment', 'sents', 'set_extension', 'similarity', 'tensor', 'text', 'text_with_ws\
33 ', 'to_array', 'to_bytes', 'to_disk', 'to_json', 'user_data', 'user_hooks', 'user_sp\
an_hooks', 'user_token_hooks', 'vector', 'vector_norm', 'vocab']
=>
```

The **to_json** method looks promising so we will import the Python pretty print library and look at the pretty printed result of calling the **to_json** method on our document stored in **doc**:

```
=> (import [pprint [pprint]])
=> (pprint (doc.to_json))
{'ents': [{'end': 21, 'label': 'PERSON', 'start': 10},
          {'end': 36, 'label': 'GPE', 'start': 30}],
 'sents': [{'end': 64, 'start': 0}],
 'text': 'President George Bush went to Mexico and he had a very good meal',
 'tokens': [{'dep': 'compound',
             'end': 9,
             'head': 2,
             'id': 0,
             'pos': 'PROPN',
             'start': 0,
             'tag': 'NNP'},
            {'dep': 'compound',
             'end': 16,
             'head': 2,
             'id': 1,
             'pos': 'PROPN',
             'start': 10,
             'tag': 'NNP'},
            {'dep': 'nsubj',
             'end': 21,
             'head': 3,
             'id': 2,
             'pos': 'PROPN',
             'start': 17,
             'tag': 'NNP'},
            {'dep': 'ROOT',
             'end': 26,
             'head': 3,
             'id': 3,
             'pos': 'VERB',
             'start': 22,
             'tag': 'VBD'},
            {'dep': 'prep',
             'end': 29,
             'head': 3,
             'id': 4,
             'pos': 'ADP',
             'start': 27,
             'tag': 'IN'},
            {'dep': 'pobj',
             'end': 36,
             'head': 4,
             'id': 5,
             'pos': 'PROPN',
             'start': 30,
             'tag': 'NNP'},
            {'dep': 'cc',
             'end': 40,
             'head': 3,
             'id': 6,
             'pos': 'CCONJ',
             'start': 37,
             'tag': 'CC'},
            {'dep': 'nsubj',
             'end': 43,
             'head': 8,
             'id': 7,
             'pos': 'PRON',
             'start': 41,
             'tag': 'PRP'},
            {'dep': 'conj',
             'end': 47,
             'head': 3,
             'id': 8,
             'pos': 'VERB',
             'start': 44,
             'tag': 'VBD'},
            {'dep': 'det',
             'end': 49,
             'head': 12,
             'id': 9,
             'pos': 'DET',
             'start': 48,
             'tag': 'DT'},
            {'dep': 'advmod',
             'end': 54,
             'head': 11,
             'id': 10,
116              'pos': 'ADV',
117              'start': 50,
118              'tag': 'RB'},
119             {'dep': 'amod',
120              'end': 59,
121              'head': 12,
122              'id': 11,
123              'pos': 'ADJ',
124              'start': 55,
125              'tag': 'JJ'},
126             {'dep': 'dobj',
127              'end': 64,
128              'head': 8,
129              'id': 12,
130              'pos': 'NOUN',
131              'start': 60,
132              'tag': 'NN'}]}
133 =>
```

The JSON data is nested dictionaries. In a later chapter on Knowledge Graphs, we will want to get the named entities like people, organizations, etc., from text and use this information to automatically generate data for Knowledge Graphs. The values for the key **ents** (stands for "entities") will be useful. Notice that the words in the original text are specified by beginning and ending text token indices (values of **head** and **end** in lines 52 to 142).

The values for the key **tokens** listed on lines 42-132 contains the head (or starting index, ending index, the token number (**id**), and the part of speech (**pos**). We will list what the parts of speech mean later.

We would like the words for each entity to be concatenated into a single string for each entity and we do this here in lines 136-137 and see the results in lines 138-139.

I like to add the entity name strings back into the dictionary representing a document and line 140 shows the use of **lfor** to create a list of lists where the sublists contain the entity name as a single string and the type of entity. We list the entity types supported by spaCy in the next section.

```
=> doc.ents
(George Bush, Mexico)
=> (for [entity doc.ents]
... (print "entity text:" entity.text "entity label:" entity.label_))
entity text: George Bush entity label: PERSON
entity text: Mexico entity label: GPE
=> (lfor entity doc.ents [entity.text entity.label_])
[['George Bush', 'PERSON'], ['Mexico', 'GPE']]
=>
```

We can also access each sentence as a separate string. In this example the original text used to create our sample document had only a single sentence so the **sents** property returns a list containing a single string:

```
=> (list doc.sents)
[President George Bush went to Mexico and he had a very good meal]
=>
```

The last example showing how to use a spaCy document object is listing each word with its part of speech:

```
=> (for [word doc]
... (print word.text word.pos_))
President PROPN
George PROPN
Bush PROPN
went VERB
to ADP
Mexico PROPN
and CCONJ
he PRON
had VERB
a DET
very ADV
good ADJ
meal NOUN
=>
```

The following list shows the definitions for the part of speech (POS) tags:

- ADJ: adjective
- ADP: adposition

- ADV: adverb
- AUX: auxiliary verb
- CONJ: coordinating conjunction
- DET: determiner
- INTJ: interjection
- NOUN: noun
- NUM: numeral
- PART: particle
- PRON: pronoun
- PROPN: proper noun
- PUNCT: punctuation
- SCONJ: subordinating conjunction
- SYM: symbol
- VERB: verb
- X: other

Implementing a HyNLP Wrapper for the Python spaCy Library

We will generate two libraries (in files **nlp_lib.hy** and **coref_nlp_lib.hy**). The first is a general NLP library and the second specifically solves the anaphora resolution, or coreference, problem. There are test programs for each library in the files **nlp_example.hy** and **coref_example.hy**.

For an example in a later chapter, we will use the library developed here to automatically generate Knowledge Graphs from text data. We will need the ability to find person, company, location, etc. names in text. We use spaCy here to do this. The types of named entities on which spaCy is pre-trained includes:

- CARDINAL: any number that is not identified as a more specific type, like money, time, etc.
- DATE
- FAC: facilities like highways, bridges, airports, etc.
- GPE: Countries, states (or provinces), and cities
- LOC: any non-GPE location
- PRODUCT
- EVENT
- LANGUAGE: any named language
- MONEY: any monetary value or unit of money
- NORP: nationalities or religious groups
- ORG: any organization like a company, non-profit, school, etc.
- PERCENT: any number in [0, 100] followed by the percent % character
- PERSON

- ORDINAL: any number spelled out, like "one", "two", etc.
- TIME

Listing for hy-lisp-python/nlp/nlp_lib.hy:

```
(import spacy)

(setv nlp-model (spacy.load "en"))

(defn nlp [some-text]
  (setv doc (nlp-model some-text))
  (setv entities (lfor entity doc.ents [entity.text entity.label_]))
  (setv j (doc.to_json))
  (setv (get j "entities") entities)
  j)
```

Listing for hy-lisp-python/nlp/nlp_example.hy:

```
#!/usr/bin/env hy

(import [nlp-lib [nlp]])

(print
  (nlp "President George Bush went to Mexico and he had a very good meal"))

(print
  (nlp "Lucy threw a ball to Bill and he caught it"))
```

```
Marks-MacBook:nlp $ ./nlp_example.hy
{'text': 'President George Bush went to Mexico and he had a very good meal', 'ents':\
 [{'start': 10, 'end': 21, 'label': 'PERSON'}, {'start': 30, 'end': 36, 'label': 'GP\
E'}], 'sents': [{'start': 0, 'end': 64}], 'tokens':

  ..LOTS OF OUTPUT NOT SHOWN..
```

Coreference (Anaphora Resolution)

Another common NLP task is coreference (or anaphora resolution) which is the process of resolving pronouns in text (e.g., he, she, it, etc.) with preceding proper nouns that pronouns refer to. A simple example would be translating "John ran fast and he fell" to "John ran fast and John fell." This is an easy example, but often proper nouns that pronouns refer to are in previous sentences and resolving coreference can be ambiguous and require knowledge of common word use and grammar. This problem is now handled by deep learning transfer models like BERT[37].

In addition to installing **spaCy** you also need the library **neuralcoref**. Only specific versions of **spaCy** and **neuralcoref** are compatible with each other. As of July 31, 2020 the following works to get dependencies and run the example for this section:

```
pip uninstall spacy neuralcoref
pip install spacy==2.1.3
python -m spacy download en
pip install neuralcoref==4.0.0
./coref_example.hy
```

Please note that version 2.1.3 of **spaCy** is older than the default version that pip installs. You might want to create a new Python virtual environment for this example or if you use Anaconda then use separate Anaconda environment.

Listing of coref_nlp_lib.hy contains a wrapper for spaCy's coreference model:

```
(import argparse os)
(import spacy neuralcoref)

(setv nlp2 (spacy.load "en"))
(neuralcoref.add_to_pipe nlp2)

(defn coref-nlp [some-text]
  (setv doc (nlp2 some-text))
  { "corefs" doc._.coref_resolved
    "clusters" doc._.coref_clusters
    "scores" doc._.coref_scores})
```

Listing of **coref_example.hy** shows code to test the Hy spaCy and coreference wrapper:

[37] https://github.com/google-research/bert

```
#!/usr/bin/env hy

(import [coref-nlp-lib [coref-nlp]])

;; tests:
(print (coref-nlp "President George Bush went to Mexico and he had a very good meal"\
))
(print (coref-nlp "Lucy threw a ball to Bill and he caught it"))
```

The output will look like:

```
Marks-MacBook:nlp $ ./coref_example.hy
{'corefs': 'President George Bush went to Mexico and President George Bush had a ver\
y good meal', 'clusters': [President George Bush: [President George Bush, he]], 'sco\
res': {President George Bush: {President George Bush: 1.5810412168502808}, George Bu\
sh: {George Bush: 4.11817741394043, President George Bush: -1.546141266822815}, Mexi\
co: {Mexico: 1.4138349294662476, President George Bush: -4.650205612182617, George B\
ush: -3.666614532470703}, he: {he: -0.5704692006111145, President George Bush: 9.385\
97583770752, George Bush: -1.4178757667541504, Mexico: -3.6565260887145996}, a very \
good meal: {a very good meal: 1.652894377708435, President George Bush: -2.554375886\
9171143, George Bush: -2.13267183303833, Mexico: -1.6889561414718628, he: -2.7667927\
742004395}}}

{'corefs': 'Lucy threw a ball to Bill and Bill caught a ball', 'clusters': [a ball: \
[a ball, it], Bill: [Bill, he]], 'scores': {Lucy: {Lucy: 0.41820740699768066}, a bal\
l: {a ball: 1.8033190965652466, Lucy: -2.721518039703369}, Bill: {Bill: 1.5611814260\
482788, Lucy: -2.8222298622131348, a ball: -1.806389570236206}, he: {he: -0.57600766\
42036438, Lucy: 3.054243326187134, a ball: -1.818403720855713, Bill: 3.0774276256561\
28}, it: {it: -1.0269954204559326, Lucy: -3.4972281455993652, a ball: -0.31290221214\
294434, Bill: -2.5343685150146484, he: -3.6687228679656982}}}
```

Anaphora resolution, also called coreference, refers to two or more words or phrases in an input text refer to the same noun. This analysis usually entails identifying which noun phrases that pronouns refer to.

Wrap-up

I spent several years of development time during the period from 1984 through 2015 working on natural language processing technology and as a personal side project I sold commercial NLP libraries that I wrote on my own time in Ruby and Common Lisp. The state-of-the-art of Deep Learning enhanced NLP is very good and the open source spaCy library makes excellent use of both conventional NLP technology and pre-trained Deep Learning models. I no longer spend very much time writing my own NLP libraries and instead use spaCy.

I urge you to read through the spaCy documentation[38] because we covered just basic functionality here that we will also need in the later chapter on automatically generating data for Knowledge Graphs. After working through the interactive REPL sessions and the examples in this chapter, you should be able to translate any Python API example code to Hy.

[38]https://spacy.io/api/doc

Datastores

I use flat files and the PostgreSQL relational database for most data storage and processing needs in my consulting business over the last twenty years. For work on large data projects at Compass Labs and Google I used Hadoop and Big Table. I will not cover big data datastores here, rather I will concentrate on what I think of as "laptop development" requirements: a modest amount of data and optimizing speed of development and ease of infrastructure setup. We will cover three datastores:

- Sqlite single-file-based relational database
- PostgreSQL relational database
- RDF library **rdflib** that is useful for semantic web and linked data applications

For graph data we will stick with RDF because it is a fairly widely used standard. Google, Microsoft, Yahoo and Yandex support schema.org[39] for defining schemas for structured data on the web. In the next chapter we will go into more details on RDF, here we look at the "plumbing" for using the **rdflib** library to manipulate and query RDF data and how to export RDF data in several formats. Then in a later chapter, we will develop tools to automatically generate RDF data from raw text as a tool for developing customized Knowledge Graphs.

In one of my previous previous books Loving Common Lisp, or the Savvy Programmer's Secret Weapon[40] I also covered the general purpose graph database Neo4j which I like to use for some use cases, but for the purposes of this book we stick with RDF.

Sqlite

We will cover two relational databases: Sqlite and PostgreSQL. Sqlite is an embedded database. There are Sqlite libraries for many programming languages and here we use the Python library.

The following examples are simple but sufficient to show you how to open a single file Sqlite database, add data, modify data, query data, and delete data. I assume that you have some familiarity with relational databases, especially concepts like data columns and rows, and SQL queries.

Let's start with putting common code for using Sqlite into a reusable library in the file **sqlite_lib.hy**:

[39]https://schema.org/
[40]https://leanpub.com/lovinglisp

```
1  (import [sqlite3 [connect version Error ]])
2
3  (defn create-db [db-file-path] ;; db-file-path can also be ":memory:"
4    (setv conn (connect db-file-path))
5    (print version)
6    (conn.close))
7
8  (defn connection [db-file-path] ;; db-file-path can also be ":memory:"
9    (connect db-file-path))
10
11 (defn query [conn sql &optional variable-bindings]
12   (setv cur (conn.cursor))
13   (if variable-bindings
14     (cur.execute sql variable-bindings)
15     (cur.execute sql))
16   (cur.fetchall))
```

The function **create-db** in lines 3-6 creates a database from a file path if it does not already exist. The function **connection** (lines 8-9) creates a persistent connection to a database defined by a file path to the single file used for a Sqlite database. This connection can be reused. The function **query** (lines 11-16) requires a connection object and a SQL query represented as a string, makes a database query, and returns all matching data in nested lists.

The following listing of file **sqlite_example.hy**shows how to use this simple library:

```
#!/usr/bin/env hy

(import [sqlite-lib [create-db connection query]])

(defn test_sqlite-lib []
  (setv dbpath ":memory:")
  (create-db dbpath)
  (setv conn (connection ":memory:"))
  (query conn "CREATE TABLE people (name TEXT, email TEXT);")
  (print
    (query conn "INSERT INTO people VALUES ('Mark', 'mark@markwatson.com')"))
  (print
    (query conn "INSERT INTO people VALUES ('Kiddo', 'kiddo@markwatson.com')"))
  (print
    (query conn "SELECT * FROM people"))
  (print
    (query conn "UPDATE people SET name = ? WHERE email = ?"
      ["Mark Watson" "mark@markwatson.com"]))
19    (print
20      (query conn "SELECT * FROM people"))
21    (print
22      (query conn "DELETE FROM people  WHERE name=?" ["Kiddo"]))
23      (print
24      (query conn "SELECT * FROM people"))
25    (conn.close))
26
27  (test_sqlite-lib)
```

We opened an in-memory database in lines 7 and 8 but we could have also created a persistent database on disk using, for example, “test_database.db” instead of **:memory**. In line 9 we create a database table with just two columns, each column holding string values.

In lines 15, 20, and 24 we are using a wild card query using the asterisk character to return all column values for each matched row in the database.

Running the example program produces the following output:

```
1  $ ./sqlite_example.hy
2  2.6.0
3  []
4  []
5  [('Mark', 'mark@markwatson.com'), ('Kiddo', 'kiddo@markwatson.com')]
6  []
7  [('Mark Watson', 'mark@markwatson.com'), ('Kiddo', 'kiddo@markwatson.com')]
8  []
9  [('Mark Watson', 'mark@markwatson.com')]
```

Line 2 shows the version of SQlist we are using. The lists in lines 1-2, 4, and 6 are empty because the functions to create a table, insert data into a table, update a row in a table, and delete rows do not return values.

In the next section we will see how PostgreSQL treats JSON data as a native data type. For sqlite, you can store JSON data as a “dumped” string value but you can’t query by key/value pairs in the data. You can encode JSON as a string and then decode it back to JSON (or as a dictionary) using:

```
(import [json [dumps loads]])

(setv json-data .....)
(setv s-data (json.dumps json-data))
(setv restored-json-data (json.loads s-data))
```

PostgreSQL

We just saw use cases for the Sqlite embedded database. Now we look at my favorite general purpose database, PostgreSQL. The PostgreSQL database server is available as a managed service on most cloud providers and it is easy to also run a PostgreSQL server on your laptop or on a VPS or server.

We will use the psycopg[41] PostgreSQL adapter that is compatible with CPython and can be installed using:

```
    pip install psycopg2
```

The following material is self-contained but before using PostgreSQL and psycopg in your own applications I recommend that you reference the psycopg documentation.

Notes for Using PostgreSQL and Setting Up an Example Database "hybook" on macOS and Linux

The following two sections may help you get PostgreSQL set up on macOS and Linux.

macOS

For macOS we use the PostgreSQL application and we will start by using the **postgres** command line utility to create a new database and table in this database. Using **postgres** account, create a new database **hybook**:

```
Marks-MacBook:datastores $ psql -d "postgres"
psql (9.6.3)
Type "help" for help.

postgres=# \d
No relations found.
postgres=# CREATE DATABASE hybook;
CREATE DATABASE
postgres=# \q
Marks-MacBook:datastores $
```

Create a table **news** in database **hybook**:

[41] http://initd.org/psycopg/

```
markw $ psql -d "hybook"
psql (9.6.3)
Type "help" for help.

hybook=# CREATE TABLE news (uri VARCHAR(50) not null, title VARCHAR(50), articletext\
 VARCHAR(500), nlpdata VARCHAR(50));
CREATE TABLE
hybook=#
```

Linux

For **Ubuntu Linux** first install PostgreSQL and then use **sudo** to use the account **postgres**:

To start a local server:

```
sudo su - postgres
/usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/main -l logfile start
```

and to stop the server:

```
sudo su - postgres
/usr/lib/postgresql/10/bin/pg_ctl -D /var/lib/postgresql/10/main -l logfile stop
```

When the PostgreSQL server is running we can use the **psql** command line program:

```
sudo su - postgres
psql

postgres@pop-os:~$ psql -d "hybook"
psql (10.7 (Ubuntu 10.7-0ubuntu0.18.10.1))
Type "help" for help.

hybook=# CREATE TABLE news (uri VARCHAR(50) not null, title VARCHAR(50), articletext\
 VARCHAR(500), nlpdata VARCHAR(50));
CREATE TABLE
hybook=# \d
        List of relations
 Schema | Name | Type  |  Owner
--------+------+-------+----------
 public | news | table | postgres
(1 row)
```

Using Hy with PostgreSQL

When using Hy (or any other Lisp language and also for Haskell), I usually start both coding and experimenting with new libraries and APIs in a REPL. Let's do that here to see from a high level how we can use psycopg on the table **news** in the database **hybook** that we created in the last section:

```
 1  Marks-MacBook:datastores $ hy
 2  hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
 3  => (import json psycopg2)
 4  => (setv conn (psycopg2.connect :dbname "hybook" :user "markw"))
 5  => (setv cur (conn.cursor))
 6  => (cur.execute "INSERT INTO news VALUES (%s, %s, %s, %s)"
 7         ["http://knowledgebooks.com/schema" "test schema"
 8         "text in article" (json.dumps {"type" "news"})])
 9  => (conn.commit)
10  => (cur.execute "SELECT * FROM news")
11  => (for [record cur]
12  ... (print record))
13  ('http://knowledgebooks.com/schema', 'test schema', 'text in article',
14   '{"type": "news"}')
15  => (cur.execute "SELECT nlpdata FROM news")
16  => (for [record cur]
17  ... (print record))
18  ('{"type": "news"}',)
19  => (cur.execute "SELECT nlpdata FROM news")
20  => (for [record cur]
21  ... (print (json.loads (first record))))
22  {'type': 'news'}
23  =>
```

In lines 6-8 and 13-14 you notice that I am using PostgreSQL's native JSON support.

As with most of the material in this book, I hope that you have a Hy REPL open and are experimenting with the APIs and code in the book's interactive REPL examples.

The file **postgres_lib.hy** wraps commonly used functionality for accessing a database, adding, modifying, and querying data in a short reusable library:

```
 1  (import [psycopg2 [connect]])
 2
 3  (defn connection-and-cursor [dbname username]
 4    (setv conn (connect :dbname dbname :user username))
 5    (setv cursor (conn.cursor))
 6    [conn cursor])
 7
 8  (defn query [cursor sql &optional variable-bindings]
 9    (if variable-bindings
10      (cursor.execute sql variable-bindings)
11      (cursor.execute sql)))
```

The function **query** in lines 8-11 executes any SQL comands so in addition to querying a database, it can also be used with appropriate SQL commands to delete rows, update rows, and create and destroy tables.

The following file **postgres_example.hy** contains examples for using the library we just defined:

```
#!/usr/bin/env hy

(import [postgres-lib [connection-and-cursor query]])

(defn test-postgres-lib []
  (setv [conn cursor] (connection-and-cursor "hybook" "markw"))
  (query cursor "CREATE TABLE people (name TEXT, email TEXT);")
  (conn.commit)
  (query cursor "INSERT INTO people VALUES ('Mark', 'mark@markwatson.com')")
  (query cursor "INSERT INTO people VALUES ('Kiddo', 'kiddo@markwatson.com')")
  (conn.commit)
  (query cursor "SELECT * FROM people")
  (print (cursor.fetchall))
  (query cursor "UPDATE people SET name = %s WHERE email = %s"
      ["Mark Watson" "mark@markwatson.com"])
  (query cursor "SELECT * FROM people")
  (print (cursor.fetchall))
  (query cursor "DELETE FROM people  WHERE name = %s" ["Kiddo"])
  (query cursor "SELECT * FROM people")
  (print (cursor.fetchall))
  (query cursor "DROP TABLE people;")
  (conn.commit)
  (conn.close))

(test-postgres-lib)
```

Here is the output from this example Hy script:

```
Marks-MacBook:datastores $ ./postgres_example.hy
[('Mark', 'mark@markwatson.com'), ('Kiddo', 'kiddo@markwatson.com')]
[('Kiddo', 'kiddo@markwatson.com'), ('Mark Watson', 'mark@markwatson.com')]
[('Mark Watson', 'mark@markwatson.com')]
```

I use PostgreSQL more than any other datastore and taking the time to learn how to manage PostgreSQL servers and write application software will save you time and effort when you are prototyping new ideas or developing data oriented product at work. I love using PostgreSQL and personally, I only use Sqlite for very small database tasks or applications.

RDF Data Using the “rdflib” Library

While the last two sections on Sqlite and PostgreSQL provided examples that you are likely to use in your own work, we will now turn to something more esoteric but still useful, the RDF notations for using data schema and RDF triple graph data in semantic web and linked data applications. I used graph databases working with Google’s Knowledge Graph when I worked there and I have had several consulting projects using linked data. You will need to understand the material in this section for the two chapters that take a deeper dive into the semantic web and linked data and also develop an example that automatically creates Knowledge Graphs.

In my work I use RDF as a notation for graph data, RDFS (RDF Schema) to define formally data types and relationship types in RDF data, and occasionally OWL (Web Ontology Language) for reasoning about RDF data and inferring new graph triple data from data explicitly defined. Here we will only cover RDF since it is the most practical linked data tool and I refer you to my other semantic web books for deeper coverage of RDF as well as RDFS and OWL.

We will go into some detail on using semantic web and linked data resources in the next chapter. Here we will study the use of library **rdflib** as a data store, reading RDF data from disk and from web resources, adding RDF statements (which are triples containing a subject, predicate, and object) and for serializing an in-memory graph to a file in one of the standard RDF XML, turtle, or NT formats.

The following REPL session shows importing the **rdflib** library, fetching RDF (in XML format) from my personal web site, printing out the triples in the graph in NT format, and showing how the graph can be queried. I added most of this RDF to my web site in 2005, with a few updates since then. The following REPL session is split up into several listings (with some long output removed) so I can explain how the **rdflib** is being used. In the first REPL listing I load an RDF file in XML format from my web site and print it in NT format. NT format can have either subject/predicate/object all on one line separated by spaces and terminated by a period or as shown below, the subject is on one line with predicate and objects printed indented on two additional lines. In both cases a period character “.” is used to terminate search RDF NT statement. The statements are displayed in arbitrary order.

```
Marks-MacBook:datastores $ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import [rdflib [Graph]])
=> (setv graph (Graph))
=> (graph.load "http://markwatson.com/index.rdf")
=> (for [[subject predicate object] graph]
... (print subject "\n   " predicate "\n   " object " ."))
http://markwatson.com/index.rdf#mark_watson_consulting_services
   http://www.w3.org/1999/02/22-rdf-syntax-ns#label
   Mark Watson Consulting Services  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#firstName
   Mark  .
http://markwatson.com/index.rdf#mark_watson
   http://www.ontoweb.org/ontology/1#name
   Mark Watson  .
http://www.markwatson.com/
   http://purl.org/dc/elements/1.1/language
   en-us  .
http://markwatson.com/index.rdf#mark_watson
   http://www.ontoweb.org/ontology/1#researchTopic
   Semantic Web  .
http://www.markwatson.com/
   http://purl.org/dc/elements/1.1/date
   2005-7-10  .
http://markwatson.com/index.rdf#mark_watson
   http://www.ontoweb.org/ontology/1#researchTopic
   RDF and RDF Schema  .
http://markwatson.com/index.rdf#mark_watson
   http://www.ontoweb.org/ontology/1#researchTopic
   ontologies  .
http://www.markwatson.com/
   http://purl.org/dc/elements/1.1/title
   Mark Watson's Home Page  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#mailbox
   mailto:markw@markwatson.com  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#homepage
   http://www.markwatson.com/  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#fullName
   Mark Watson  .
```

```
http://markwatson.com/index.rdf#mark_watson_consulting_services
   http://www.ontoweb.org/ontology/1#name
   Mark Watson Consulting Services  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#company
   Mark Watson Consulting Services  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   http://www.w3.org/2000/10/swap/pim/contact#Person  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/1999/02/22-rdf-syntax-ns#value
   Mark Watson  .
http://www.markwatson.com/
   http://purl.org/dc/elements/1.1/creator
   http://markwatson.com/index.rdf#mark_watson  .
http://www.markwatson.com/
   http://purl.org/dc/elements/1.1/description

      Mark Watson is the author of 16 published books and a consultant specializing \
in artificial intelligence and Java technologies.
      .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   http://www.ontoweb.org/ontology/1#Person  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/2000/10/swap/pim/contact#motherTongue
   en  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   http://markwatson.com/index.rdf#Consultant  .
http://markwatson.com/index.rdf#mark_watson_consulting_services
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   http://www.ontoweb.org/ontology/1#Organization  .
http://markwatson.com/index.rdf#mark_watson
   http://www.w3.org/1999/02/22-rdf-syntax-ns#label
   Mark Watson  .
http://markwatson.com/index.rdf#Sun_ONE
   http://www.w3.org/1999/02/22-rdf-syntax-ns#type
   http://www.ontoweb.org/ontology/1#Book  .
=>
```

We will cover the SPARQL query language in more detail in the next chapter but for now, notice that SPARQL is similar to SQL queries. SPARQL queries can find triples in a graph matching simple patterns, match complex patterns, and update and delete triples in a graph. The following simple SPARQL query finds all triples with the predicate equal to http://www.w3.org/2000/10/swap/pim/contact#company and prints out the subject and object of any matching triples:

```
=> (for [[subject object
... (graph.query
...   "select ?s ?o where { ?s <http://www.w3.org/2000/10/swap/pim/contact#company> \
?o }")]
... (print subject "\n contact company: " object))
http://markwatson.com/index.rdf#mark_watson
  contact company:  Mark Watson Consulting Services
```

We will see more examples of the SPARQL query language in the next chapter. For now, notice that the general form of a **select** query statement is a list of query variables (names beginning with a question mark) and a **where** clause in curly brackets that contains matching patterns. This SPARQL query is simple, but like SQL queries, SPARQL queries can get very complex. I only lightly cover SPARQL in this book. You can get PDF copies of my two older semantic web books for free: Practical Semantic Web and Linked Data Applications, Java, Scala, Clojure, and JRuby Edition[42] and Practical Semantic Web and Linked Data Applications, Common Lisp Edition[43]. There are links to relevant git repos and other information on my book web page[44].

As I mentioned, the RDF data on my web site has been essentially unchanged since 2005. What if I wanted to update it noting that more recently I worked as a contractor at Google and as a Deep Learning engineering manager at Capital One? In the following listing, continuing the same REPL session, I will add two RDF statements indicating additional jobs and show how to serialize the new updated graph in three formats: XML, turtle (my favorite RDF notation) and NT:

```
=> (import rdflib)
=> (setv mark-node (rdflib.URIRef  "http://markwatson.com/index.rdf#mark_watson"))
=> mark-node
rdflib.term.URIRef('http://markwatson.com/index.rdf#mark_watson')
=> (setv company-node (rdflib.URIRef "http://www.w3.org/2000/10/swap/pim/contact#com\
pany"))
=> (graph.add [mark-node company-node (rdflib.Literal "Google")])
=> (graph.add [mark-node company-node (rdflib.Literal "Capital One")])
=> (for [[subject object]
...       (graph.query
...         "select ?s ?o where { ?s <http://www.w3.org/2000/10/swap/pim/contact#com\
pany> ?o }")]
... (print subject " contact company: " object))
http://markwatson.com/index.rdf#mark_watson  contact company:  Mark Watson Consultin\
```

[42]https://markwatson.com/opencontentdata/book_java.pdf
[43]https://markwatson.com/opencontentdata/book_lisp.pdf
[44]https://markwatson.com/books/

```
103 g Services
104 http://markwatson.com/index.rdf#mark_watson  contact company:  Google
105 http://markwatson.com/index.rdf#mark_watson  contact company:  Capital One
106 =>
107 => (graph.serialize :format "pretty-xml")
108 <?xml version="1.0" encoding="utf-8"?>
109 <rdf:RDF\n  xmlns:dc="http://purl.org/dc/elements/1.1/"
110   xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#"
111   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
112   xmlns:ow="http://www.ontoweb.org/ontology/1#"
113   xmlns:ns1="http://markwatson.com/index.rdf#">
114 
115     LOTS OF STUFF NOT SHOWN
116 
117 </rdf:Description>\n</rdf:RDF>
```

I like Turtle RDF notation better than the XML notation because Turtle is easier to read and understand. Here, on line 118 we serialize the graph (with new nodes added above in lines 90 to 96) to Turtle:

```
118 => (graph.serialize :format "turtle")
119 @prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
120 @prefix dc: <http://purl.org/dc/elements/1.1/> .
121 @prefix ns1: <http://markwatson.com/index.rdf#> .
122 @prefix ow: <http://www.ontoweb.org/ontology/1#> .
123 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
124 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
125 @prefix xml: <http://www.w3.org/XML/1998/namespace> .
126 @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
127 
128 ns1:mark_watson_consulting_services a ow:Organization ;
129     ow:name "Mark Watson Consulting Services" ;
130     rdf:label "Mark Watson Consulting Services" .
131 <http://www.markwatson.com/>
132     dc:creator ns1:mark_watson ;
133     dc:date "2005-7-10" ;
134     dc:description
135           "Mark Watson is the author of 16 published books and a consultant
136           specializing in artificial intelligence and Java technologies." ;
137      dc:format "text/html" ;
138      dc:language "en-us" ;
139      dc:title "Mark Watson\'s Home Page" ;
140      dc:type "Consultant Homepage" .
```

```
141 ns1:mark_watson a ns1:Consultant,
142                   ow:Person,
143                   contact:Person ;
144                 ow:name "Mark Watson" ;
145                 ow:researchTopic "RDF and RDF Schema",
146                                  "Semantic Web",
147                                  "ontologies" ;
148                 rdf:label "Mark Watson" ;
149                 rdf:value "Mark Watson" ;
150                 contact:company "Capital One",
151                                 "Google",
152                                 "Mark Watson Consulting Services" ;
153                 contact:familyName "Watson" ;
154                 contact:firstName "Mark" ;
155                 contact:fullName "Mark Watson" ;
156                 contact:homepage <http://www.markwatson.com/> ;
157                 contact:mailbox <mailto:markw@markwatson.com> ;
158                 contact:motherTongue "en" .
```

In addition to the Turtle format I also use the simpler NT format that puts URI prefixes inline and unlike Turtle does not use prefix abrieviations. Here in line 159 we serialize to NT format:

```
159 => (graph.serialize :format "nt")
160 <http://markwatson.com/index.rdf#Sun_ONE> <http://www.ontoweb.org/ontology/1#booktit\
161 le> "Sun ONE Services - J2EE" .
162 <http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/language> "en-us" .
163 <http://markwatson.com/index.rdf#Sun_ONE> <http://www.ontoweb.org/ontology/1#author>\
164  <http://markwatson.com/index.rdf#mark_watson> .
165 <http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/date> "2005-7-10" .
166 <http://markwatson.com/index.rdf#mark_watson> <http://www.ontoweb.org/ontology/1#res\
167 earchTopic> "ontologies" .
168 <http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/type> "Consultant Home\
169 page" .
170 <http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/2000/10/swap/pim/co\
171 ntact#homepage> <http://www.markwatson.com/> .
172 <http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/2000/10/swap/pim/co\
173 ntact#company> "Google" .
174 <http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/2000/10/swap/pim/co\
175 ntact#company> "Capital One" .
176 <http://markwatson.com/index.rdf#mark_watson> <http://www.ontoweb.org/ontology/1#res\
177 earchTopic> "Semantic Web" .
178 <http://markwatson.com/index.rdf#mark_watson> <http://www.ontoweb.org/ontology/1#res\
179 earchTopic> "RDF and RDF Schema" .
```

```
<http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/title> "Mark Watson\'s\
 Home Page" .
<http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/2000/10/swap/pim/co\
ntact#mailbox> <mailto:markw@markwatson.com> .
<http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/format> "text/html" .
<http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/2000/10/swap/pim/co\
ntact#fullName> "Mark Watson" .
<http://markwatson.com/index.rdf#mark_watson> <http://www.w3.org/1999/02/22-rdf-synt\
ax-ns#type> <http://www.w3.org/2000/10/swap/pim/contact#Person> .
<http://www.markwatson.com/> <http://purl.org/dc/elements/1.1/creator> <http://markw\
atson.com/index.rdf#mark_watson> .

      LOTS OF STUFF NOT SHOWN
=>
```

Using Relational Database as a Backend for rdflib

I am not going to cover using a non-default rdflib backend, but if you want to be able to load large RDF data sets and persist the data then you can use the SQLAlchemy plugin extension.

First, install the Python SQLAlchemy RDF library:

```
    pip install rdflib_sqlalchemy
```

Then, follow the test examples at the rdflib-sqlalchemy[45] github repository.

If I need to use large RDF data sets I prefer to not use rdflib and instead use SPARQL to access a free or open source standalone RDF data store like OpenLink Virtuoso[46] or GraphDB™ Free Edition[47]. I also like and recommend the commercial AllegroGraph and Stardog RDF server products.

Wrap-up

We will go into much more detail on practical uses of RDF and SPARQL in the next chapter. I hope that you worked through the REPL examples in this section because if you understand the basics of using the **rdflib** then you will have an easier time understanding the more abstract material in the next chapter.

[45]https://github.com/RDFLib/rdflib-sqlalchemy
[46]https://en.wikipedia.org/wiki/Virtuoso_Universal_Server
[47]https://www.ontotext.com/products/graphdb/graphdb-free/

Linked Data and the Semantic Web

Tim Berners Lee, James Hendler, and Ora Lassila wrote in 2001 an article for Scientific American where they introduced the term Semantic Web. Here I do not capitalize semantic web and use the similar term linked data somewhat interchangeably with semantic web.

I assume that you read the section RDF Data Using the "rdflib" Library in the last chapter.

In the same way that the web allows links between related web pages, linked data supports linking associated data on the web together. I view linked data as a relatively simple way to specify relationships between data sources on the web while the semantic web has a much larger vision: the semantic web has the potential to be the entirety of human knowledge represented as data on the web in a form that software agents can work with to answer questions, perform research, and to infer new data from existing data.

While the "web" describes information for human readers, the semantic web is meant to provide structured data for ingestion by software agents. This distinction will be clear as we compare WikiPedia, made for human readers, with DBPedia which uses the info boxes on WikiPedia topics to automatically extract RDF data describing WikiPedia topics. Let's look at the WikiPedia topic for the town I live in, Sedona Arizona, and show how the info box on the English version of the WikiPedia topic page for Sedona https://en.wikipedia.org/wiki/Sedona,_Arizona[48] maps to the DBPedia page http://dbpedia.org/page/Sedona,_Arizona[49]. Please open both of these WikiPedia and DBPedia URIs in two browser tabs and keep them open for reference.

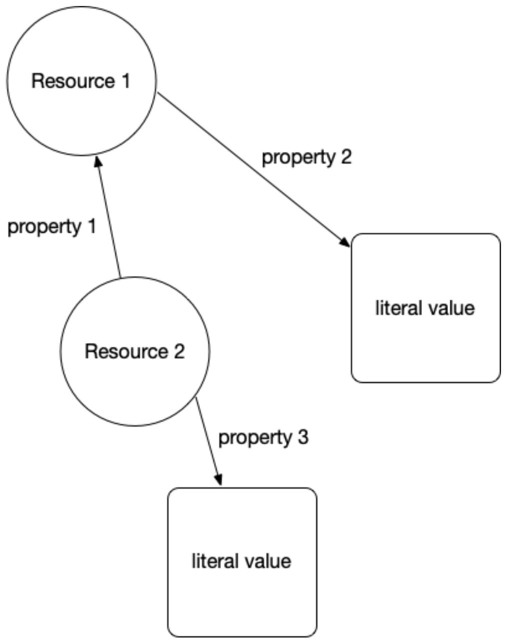
I assume that the format of the WikiPedia page is familiar so let's look at the DBPedia page for Sedona that in human readable form shows the RDF statements with Sedona Arizona as the subject. RDF is used to model and represent data. RDF is defined by three values so an instance of an RDF statement is called a *triple* with three parts:

- subject: a URI (also referred to as a "Resource")
- property: a URI (also referred to as a "Resource")
- value: a URI (also referred to as a "Resource") or a literal value (like a string)

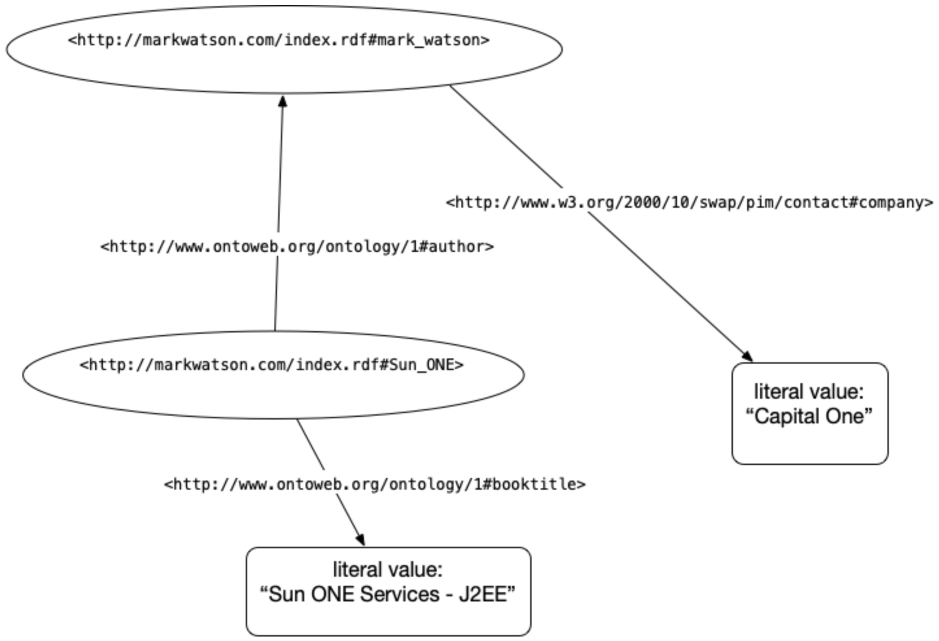
The subject for each Sedona related triple is the above URI for the DBPedia human readable page. The subject and property references in an RDF triple will almost always be a URI that can ground an entity to information on the web. The human readable page for Sedona lists several properties and the values of these properties. One of the properties is "dbo:areaCode" where "dbo" is a name space reference (in this case for a DatatypeProperty[50]).

[48]https://en.wikipedia.org/wiki/Sedona,_Arizona
[49]http://dbpedia.org/page/Sedona,_Arizona
[50]http://www.w3.org/2002/07/owl#DatatypeProperty

The following two figures show an abstract representation of linked data and then a sample of linked data with actual web URIs for resources and properties:



Abstract RDF representation with 2 Resources, 2 literal values, and 3 Properties


<http://markwatson.com/index.rdf#mark_watson>
<http://www.w3.org/2000/10/swap/pim/contact#company>
<http://www.ontoweb.org/ontology/1#author>
<http://markwatson.com/index.rdf#Sun_ONE>
literal value: "Capital One"
<http://www.ontoweb.org/ontology/1#booktitle>
literal value: "Sun ONE Services - J2EE"

Concrete example using RDF seen in last chapter showing the RDF representation with 2 Resources, 2 literal values, and 3 Properties

We saw a SPARQL Query (SPARQL for RDF data is similar to SQL for relational database queries) in the last chapter. Let's look at another example using the RDF in the last figure:

```
select ?v where {  <http://markwatson.com/index.rdf#Sun_ONE>
                   <http://www.ontoweb.org/ontology/1#booktitle>
                   ?v }
```

This query should return the result “Sun ONE Services - J2EE”. If you wanted to query for all URI resources that are books with the literal value of their titles, then you can use:

```
select ?s ?v where {  ?s
                      <http://www.ontoweb.org/ontology/1#booktitle>
                      ?v }
```

Note that **?s** and **?v** are arbitrary query variable names, here standing for “subject” and “value”. You can use more descriptive variable names like:

```
select ?bookURI ?bookTitle where
    { ?bookURI
      <http://www.ontoweb.org/ontology/1#booktitle>
      ?bookTitle }
```

We will be diving a little deeper into RDF examples in the next chapter when we write a tool for generating RDF data from raw text input. For now I want you to understand the idea of RDF statements represented as triples, that web URIs represent things, properties, and sometimes values, and that URIs can be followed manually (often called “dereferencing”) to see what they reference in human readable form.

Understanding the Resource Description Framework (RDF)

Text data on the web has some structure in the form of HTML elements like headers, page titles, anchor links, etc. but this structure is too imprecise for general use by software agents. RDF is a method for encoding structured data in a more precise way.

We used the RDF data on my web site in the last chapter to introduce the “plumbing” of using the **rdflib** Python library to access, manipulate, and query RDF data.

Resource Namespaces Provided in rdflib

The following standard namespaces are predefined in **rdflib**:

- RDF https://www.w3.org/TR/rdf-syntax-grammar/[51]

[51]https://www.w3.org/TR/rdf-syntax-grammar/

- RDFS https://www.w3.org/TR/rdf-schema/[52]
- OWL http://www.w3.org/2002/07/owl#[53]
- XSD http://www.w3.org/2001/XMLSchema#[54]
- FOAF http://xmlns.com/foaf/0.1/[55]
- SKOS http://www.w3.org/2004/02/skos/core#[56]
- DOAP http://usefulinc.com/ns/doap#[57]
- DC http://purl.org/dc/elements/1.1/[58]
- DCTERMS http://purl.org/dc/terms/[59]
- VOID http://rdfs.org/ns/void#[60]

Let's look into the Friend of a Friend (FOAF) namespace. Click on the above link for FOAF http://xmlns.com/foaf/0.1/[61] and find the definitions for the FOAF Core:

```
    Agent
    Person
    name
    title
    img
    depiction (depicts)
    familyName
    givenName
    knows
    based_near
    age
    made (maker)
    primaryTopic (primaryTopicOf)
    Project
    Organization
    Group
    member
    Document
    Image
```

and for the Social Web:

[52]https://www.w3.org/TR/rdf-schema/
[53]http://www.w3.org/2002/07/owl#
[54]http://www.w3.org/2001/XMLSchema#
[55]http://xmlns.com/foaf/0.1/
[56]http://www.w3.org/2004/02/skos/core#
[57]http://usefulinc.com/ns/doap#
[58]http://purl.org/dc/elements/1.1/
[59]http://purl.org/dc/terms/
[60]http://rdfs.org/ns/void#
[61]http://xmlns.com/foaf/0.1/

```
nick
mbox
homepage
weblog
openid
jabberID
mbox_sha1sum
interest
topic_interest
topic (page)
workplaceHomepage
workInfoHomepage
schoolHomepage
publications
currentProject
pastProject
account
OnlineAccount
accountName
accountServiceHomepage
PersonalProfileDocument
tipjar
sha1
thumbnail
logo
```

You now have seen a few common Schemas for RDF data. Another Schema that is widely used for annotating web sites, that we won't need for our examples here, is schema.org[62]. Let's now use a Hy REPL session to explore namespaces and programatically create RDF using **rdflib**:

```
Marks-MacBook:database $ hy
hy 0.17.0+108.g919a77e using CPython(default) 3.7.3 on Darwin
=> (import [rdflib.namespace [FOAF]])
=> FOAF
Namespace('http://xmlns.com/foaf/0.1/')
=> FOAF.name
rdflib.term.URIRef('http://xmlns.com/foaf/0.1/name')
=> FOAF.title
rdflib.term.URIRef('http://xmlns.com/foaf/0.1/title')
=> (import rdflib)
=> (setv graph (rdflib.Graph))
```

[62]https://schema.org

```
=> (setv mark (rdflib.BNode))
=> (graph.bind "foaf" FOAF)
=> (import [rdflib [RDF]])
=> (graph.add [mark RDF.type FOAF.Person])
=> (graph.add [mark FOAF.nick (rdflib.Literal "Mark" :lang "en")])
=> (graph.add [mark FOAF.name (rdflib.Literal "Mark Watson" :lang "en")])
=> (for [node graph] (print node))
(rdflib.term.BNode('N21c7fa7385b545eb8a7e3821b7cb5'), rdflib.term.URIRef('http://www\
.w3.org/1999/02/22-rdf-syntax-ns#type'), rdflib.term.URIRef('http://xmlns.com/foaf/0\
.1/Person'))
(rdflib.term.BNode('N21c7fa7385b545eb8a7e3821b7cb5'), rdflib.term.URIRef('http://xml\
ns.com/foaf/0.1/name'), rdflib.term.Literal('Mark Watson', lang='en'))
(rdflib.term.BNode('N21c7fa7385b545eb8a7e3821b7cb5'), rdflib.term.URIRef('http://xml\
ns.com/foaf/0.1/nick'), rdflib.term.Literal('Mark', lang='en'))
=> (graph.serialize :format "pretty-xml")
b'<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <foaf:Person rdf:nodeID="N21c7fa7385b545eb8a7e3821b75b9cb5">
    <foaf:name xml:lang="en">Mark Watson</foaf:name>
    <foaf:nick xml:lang="en">Mark</foaf:nick>
  </foaf:Person>
</rdf:RDF>\n'
=> (graph.serialize :format "turtle")
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[] a foaf:Person ;
    foaf:name "Mark Watson"@en ;
    foaf:nick "Mark"@en .

=> (graph.serialize :format "nt")
_:N21c7fa7385b545eb8a7e3821b75b9cb5
   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   <http://xmlns.com/foaf/0.1/Person> .
_:N21c7fa7385b545eb8a7e3821b75b9cb5 <http://xmlns.com/foaf/0.1/name> "Mark Watson"@e\
n .
_:N21c7fa7385b545eb8a7e3821b75b9cb5 <http://xmlns.com/foaf/0.1/nick> "Mark"@en .
```

```
=>
```

Understanding the SPARQL Query Language

For the purposes of the material in this book, the two sample SPARQL queries here and in the last chapter are sufficient for you to get started using **rdflib** with arbitrary RDF data sources and simple queries.

The Apache Foundation has a good introduction to SPARQL[63] that I refer you to for more information.

Wrapping the Python rdflib Library

I hope that I have provided you with enough motivation to explore RDF data sources and consider the use of linked data/semantic web technologies for your projects.

You can install using the source code for **rdflib**[64] or using:

```
pip install rdflib
```

If I depend on a library, regardless of the programming language, I like to keep an up-to-date copy of the source code ready at hand. There is sometimes no substitute for having library code available to read.

In the next chapter we will use natural language processing to extract structured information from raw text and automatically generate RDF data.

[63]https://jena.apache.org/tutorials/sparql.html
[64]https://github.com/RDFLib/rdflib

Knowledge Graph Creator

A Knowledge Graph, that I often abbreviate as **KG**, is a graph database using a schema to define types (both objects and relationships between objects) and properties that link property values to objects. The term “Knowledge Graph” is both a general term and also sometimes refers to the specific Knowledge Graph used at Google which I worked with while working there in 2013. Here, we use KG to reference the general technology of storing knowledge in graph databases.

The application we develop here, the Knowledge Graph Creator (which I often refer to as KGCreator) is a utility that I use to generate small Knowledge Graphs from input text.

Knowledge engineering and knowledge representation are disciplines that started in the 1980s and are still both current research topics and used in industry. I view linked data, the semantic web, and KGs as extensions of this earlier work.

We base our work here on RDF. There is a general type of KGs that are also widely used in industry and that we will not cover here: property graphs, as used in Neo4J. Property graphs are general graphs that place no restrictions on the number of links a graph node may have and allow general data structures to be stored as node data and for the property links between nodes. Property links can have attributes, like nodes in the graph.

Semantic web data as represented by subject/property/value RDF triples are more constrained than property graphs but support powerful logic inferencing to better use data that is implicit in a graph but not explicitly stated (i.e., data is more easily inferred).

We covered RDF data in some detail in the last chapter. Here we will implement a toolset for converting unstructured text into RDF data using a few schema definitions from schema.org[65]. I believe in both the RDF and the general graph database approaches but here we will just use RDF.

Historically Knowledge Graphs used semantic web technology like Resource Description Framework (RDF)[66] and Web Ontology Language (OWL)[67]. I wrote two books in 2010 on semantic web technologies and you can get free PDFs for the Common Lisp version[68] (code is here[69]) and the Java/Clojure/Scala version[70] (code is here[71]). These free books might interest you after working through the material in this chapter.

I have an ongoing personal research project for creating knowledge graphs from various data sources. You can read more at my KGCreator web site[72]. I have simplified versions of my KGCreator software

[65]https://schema.org/
[66]https://en.wikipedia.org/wiki/Resource_Description_Framework
[67]https://en.wikipedia.org/wiki/Web_Ontology_Language
[68]http://markwatson.com/opencontentdata/book_lisp.pdf
[69]https://github.com/mark-watson/lisp_practical_semantic_web
[70]http://markwatson.com/opencontentdata/book_java.pdf
[71]https://github.com/mark-watson/java_practical_semantic_web
[72]http://www.kgcreator.com/

implemented in both my Haskell Book[73] and in my most recent Common Lisp book[74]. The example here is similar to my Common Lisp implementation, except that it is implemented in the Hy language and I only support generating RDF. The examples in my Haskell and Common Lisp books also generate data for the Neo4J graph database.

What is a KG? It is a modern way to organize and access structured data and integrate data and metadata with other automated systems.

A Knowledge Graph is different from just a graph database containing graph data. The difference is that a KG will in general use Schemas, Taxonomy's and Ontology's that define the allowed types and structure of data and allowed relationships.

There is also an executable aspect of KGs since their primary use may be to support other systems in an organization.

Recommended Industrial Use of Knowledge Graphs

Who needs a KG? How do you get started?

If people in your organization are spending much time doing general web search, it might be a signal that you should maintain your organization's curated knowledge in a human searchable and software accessible way. A possible application is an internal search engine that mixes public web search APIs with search for knowledge used internally inside your organization.

Here are a few use cases:

- At Google we used their Knowledge Graph for researching new internal systems that were built on their standard Knowledge Graph, with new schemas and data added.
- Digital transformations: start by using a KG to hold metadata for current data in already existing databases. A KG of metadata can provide you with a virtual data lake. It is common to build a large data lake and then have staff not be able to find data. Don't try to do everything at once.
- Capture and preserve senior human expertise. The act of building an Ontology for in-house knowledge helps to understand how to organize data and provides people with a common vocabulary to discuss and model business processes.
- KYC (Know Your Customer) applications using data from many diverse data sources.
- Take advantage of expertise in a domain (e.g., healthcare or financial services) to build a Taxonomy and Ontology to use to organize available data. For most domains, there are standard existing Schemas, Taxonomy's and Ontology's that can be identified and used as-is or extended for your organization.

To get started:

[73]https://leanpub.com/haskell-cookbook
[74]https://leanpub.com/lovinglisp

- Start small with just one use case.
- Design a Schema that identifies object types and relationships
- Write some acceptance test cases that you want a prototype to be able to serve as a baseline to develop against.
- Avoid having too many stakeholders in early prototype projects — try to choose stakeholders based on potential stakeholders' initial enthusiasm.

A good way to start is to identify a single problem, determine the best data sources to use, define an Ontology that is just sufficient to solve the current problem and build a prototype "vertical slice" application. Lessons learned with a quick prototype will inform you on what was valuable and what to put effort into when expanding your KG. Start small and don't try to build a huge system without taking many small development and evaluation steps.

What about KGs for small organizations? Small companies have less development resources but starting small and implementing a system that models the key data relationships, customer relationships, etc., does not require excessive resources. Just capturing where data comes from and who is responsible for maintaining important data sources can be valuable.

What about KGs for individuals? Given the effort involved in building custom KGs, one possible individual use case is developing KGs for commercial sale.

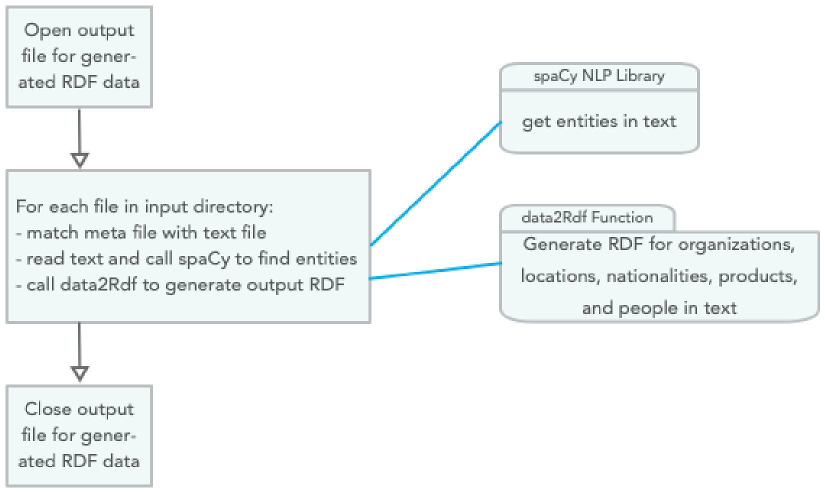
The application that we develop next is one way to quickly bootstrap a new KG by populating it with automatically generated RDF than can be manually curated by removing statements and adding new statements as appropriate.

Design of KGCreator Application

The example application developed here processes input text files in the sub-directory **test_data**. For each file with the extension **.txt** in **test_data**, there should be a matching file with the extension **.meta** that contains the origin URI for the corresponding text file. The git repository for this book has a few files in **test_data** that you can experiment with or replace with your own data:

```
$ ls test_data
test1.meta test1.txt test2.meta test2.txt test3.meta test3.txt
```

The *.txt files contain plain text for analysis and the *.meta files contain the original web source URI for the corresponding *.txt files. Using the spaCy library and Python/Hy's standard libraries for file access, the KGCreator is simple to implement. Here is the overall design of this example:


Open output file for generated RDF data
For each file in input directory:
- match meta file with text file
- read text and call spaCy to find entities
- call data2Rdf to generate output RDF
Close output file for generated RDF data
spaCy NLP Library
get entities in text
data2Rdf Function
Generate RDF for organizations, locations, nationalities, products, and people in text

Overview of the Knowledge Graph Creator script

We will develop two versions of the Knowledge Graph Creator. The first generates RDF that uses string values for the object part of generated RDF statements. The second implementation attempts to resolve these string values to DBPedia URIs.

Using only the spaCy NLP library that we used earlier and the built in Hy/Python libraries, this first example (uses strings a object values) is implemented in just 58 lines of Hy code that is seen in the following three code listings:

```
 1  #!/usr/bin/env hy
 2
 3  (import [os [scandir]])
 4  (import [os.path [splitext exists]])
 5  (import spacy)
 6
 7  (setv nlp-model (spacy.load "en"))
 8
 9  (defn find-entities-in-text [some-text]
10    (defn clean [s]
11      (.strip (.replace s "\n" " ")))
12    (setv doc (nlp-model some-text))
13    (map list (lfor entity doc.ents [(clean entity.text) entity.label_])))
```

In lines 3 and 4 we import three standard Python utilities we need for finding all files in a directory, checking to see if a file exists, and splitting text into tokens. In line 7 we load the English language spaCy model and save the value of the model in the variable **nlp-model**. The function find-entities-in-text uses the spaCy English language model to find entities like organizations, people, etc., in text and cleans entity names by removing new line characters and other unnecessary white space (nested function **clean** in lines 10 and 11). We can run a test in a REPL:

```
=> (list (find-entities-in-text "John Smith went to Los Angeles to work at IBM"))
[['John Smith', 'PERSON'], ['Los Angeles', 'GPE'], ['IBM', 'ORG']]
```

The function **find-entities-in-text** returns a map object so I wrapped the results in a **list** to print out the entities in the test sentence. The entity types used by spaCy were defined in an earlier chapter, here we just use the entity types defined in lines 21-26 in the following listing:

```
14 (defn data2Rdf [meta-data entities fout]
15   (for [[value abbreviation] entities]
16     (if (in abbreviation e2umap)
17       (.write fout (+ "<" meta-data ">\t" (get e2umap abbreviation) "\t" "\""
18                        value "\"" " .\n")))))
19 
20 (setv e2umap {
21   "ORG" "<https://schema.org/Organization>"
22   "LOC" "<https://schema.org/location>"
23   "GPE" "<https://schema.org/location>"
24   "NORP" "<https://schema.org/nationality>"
25   "PRODUCT" "<https://schema.org/Product>"
26   "PERSON" "<https://schema.org/Person>"})
```

In lines 28-39 we open an output file for writing generated RDF data and loop through all text files in the input directory and call the function **process-file** for each text + meta file pair in the input directory:

```
28 (defn process-directory [directory-name output-rdf]
29   (with [frdf (open output-rdf "w")]
30     (with [entries (scandir directory-name)]
31       (for [entry entries]
32         (setv [_ file-extension] (splitext entry.name))
33         (if (= file-extension ".txt")
34             (do
35               (setv check-file-name (+ (cut entry.path 0 -4) ".meta"))
36               (if (exists check-file-name)
37                   (process-file entry.path check-file-name frdf)
38                   (print "Warning: no .meta file for" entry.path
39                          "in directory" directory-name))))))))
```

```
(defn process-file [txt-path meta-path frdf]

  (defn read-data [text-path meta-path]
    (with [f (open text-path)] (setv t1 (.read f)))
    (with [f (open meta-path)] (setv t2 (.read f)))
    [t1 t2])

  (defn modify-entity-names [ename]
    (.replace ename "the " ""))

  (setv [txt meta] (read-data txt-path meta-path))
  (setv entities (find-entities-in-text txt))
  (setv entities ;; only operate on a few entity types
        (lfor [e t] entities
              :if (in t ["NORP" "ORG" "PRODUCT" "GPE" "PERSON" "LOC"])
              [(modify-entity-names e) t]))
  (data2Rdf meta entities frdf))

(process-directory "test_data" "output.rdf")
```

We will look at generated output, problems with it, and how to fix these problems in the next section.

Problems with using Literal Values in RDF

Using the Hy script in the last section, let's look at some of the generated RDF for the text files in the input test directory (most output is not shown). In each triple the first item, the subject, is the URI of the data source, the second item in each statement is a URI representing a relationship (or property), and the third item is a literal string value:

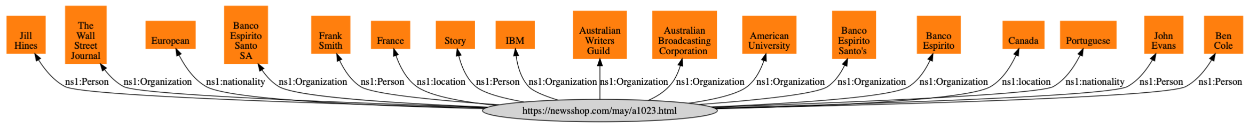
```
<https://newsshop.com/may/a1023.html>
  <https://schema.org/nationality>        "Portuguese" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "Banco Espirito Santo SA" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Person>              "John Evans" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "Banco Espirito" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "The Wall Street Journal" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "IBM" .
<https://newsshop.com/may/a1023.html>
```

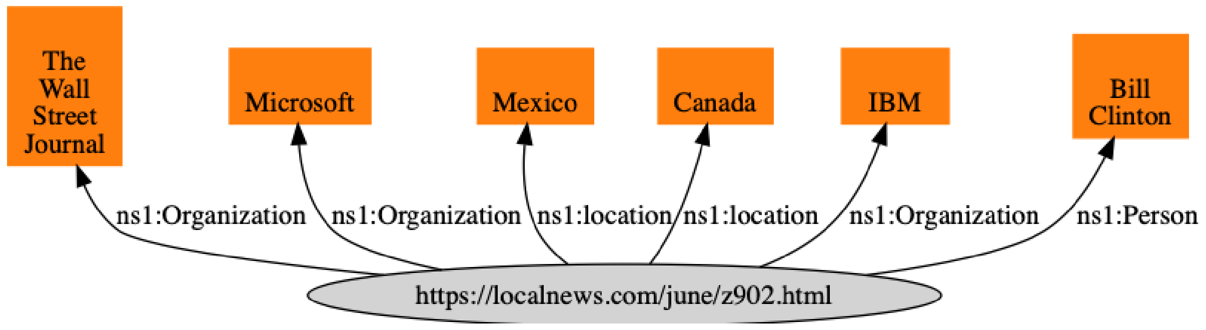
```
  <https://schema.org/location>        "Canada" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "Australian Broadcasting Corporation" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Person>        "Frank Smith" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "Australian Writers Guild" .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>        "American University" .
<https://localnews.com/june/z902.html>
  <https://schema.org/Organization>        "The Wall Street Journal" .
<https://localnews.com/june/z902.html>
  <https://schema.org/location>        "Mexico" .
<https://localnews.com/june/z902.html>
  <https://schema.org/location>        "Canada" .
<https://localnews.com/june/z902.html>
  <https://schema.org/Person>        "Bill Clinton" .
<https://localnews.com/june/z902.html>
  <https://schema.org/Organization>        "IBM" .
<https://localnews.com/june/z902.html>
  <https://schema.org/Organization>        "Microsoft" .
<https://abcnews.go.com/US/violent-long-lasting-tornadoes-threaten-oklahoma-texas/st\
ory?id=63146361>
  <https://schema.org/Person>        "Jane Deerborn" .
<https://abcnews.go.com/US/violent-long-lasting-tornadoes-threaten-oklahoma-texas/st\
ory?id=63146361>
  <https://schema.org/location>        "Texas" .
```

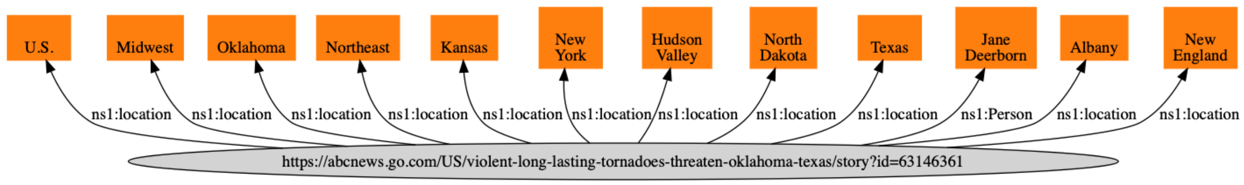
Let's visualize the results in a bash shell:

```
$ git clone https://github.com/fatestigma/ontology-visualization
$ cd ontology-visualization
$ chmod +x ontology_viz.py
$ ./ontology_viz.py -o test.dot output.rdf  -O ontology.ttl
$ # copy the file output.rdf from examples repo directory hy-lisp-python/kgcreator
$ dot -Tpng -o test.png test.dot
$ open test.png
```

Edited to fit on the page, the output looks like:







Because we used literal values, notice how for example the node for the entity **IBM** is not shared and thus a software agent using this RDF data cannot, for example, infer relationships between two news sources that both have articles about IBM. We will work on a solution to this problem in the next section.

Revisiting This Example Using URIs Instead of Literal Values

Note that in the figure in the previous section that nodes for literal values (e.g., for "IBM") are not shared. In this section we will copy the file **kgcreator.hy** to **kgcreator_uri.hy** add a few additions to map string literal values for entity names to http://dbpedia.org[75] URIs by individually searching Google using the pattern "DBPedia 'entity name'" and defining a new map **v2umap** for mapping literal values to DBPedia URIs.

Note: In a production system (not a book example), I would use https://www.wikidata.org database download[76] to download all of WikiData (which includes DBPedia data) and use a fuzzy text matching to find WikiData URIs for string literals. The compressed WikiData JSON data file is about 50 GB. Here we will manually find DBPedia for entity names that are in the example data.

In **kgcreator_uri.hy** we add a map **v2umap** for selected entity literal names to DBPedia URIs that I manually created using a web search on the DBPedia domain:

[75] http://dbpedia.org
[76] https://www.wikidata.org/wiki/Wikidata:Database_download

```
(setv v2umap { ;; object literal value to URI mapping
  "IBM" "<http://dbpedia.org/page/IBM>"
  "The Wall Street Journal" "<http://dbpedia.org/page/The_Wall_Street_Journal>"
  "Banco Espirito" "<http://dbpedia.org/page/Banco_Esp%C3%ADrito_Santo>"
  "Australian Broadcasting Corporation"
  "http://dbpedia.org/page/Australian_Broadcasting_Corporation"
  "Australian Writers Guild"
  "http://dbpedia.org/page/Australian_Broadcasting_Corporation"
  "Microsoft" "http://dbpedia.org/page/Microsoft"})
```

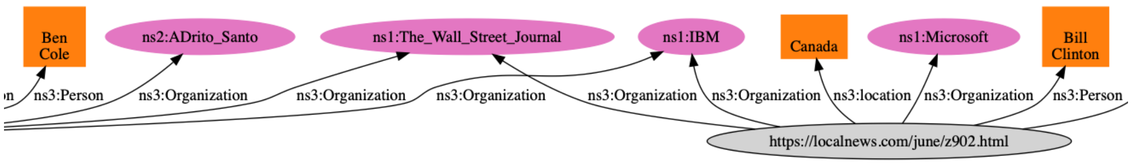
We also make a change in the function **data2Rdf** to use the map **v2umap**:

```
(defn data2Rdf [meta-data entities fout]
  (for [[value abbreviation] entities]
    (setv a-literal (+ "\"" value "\""))
    (if (in value v2umap) (setv a-literal (get v2umap value)))
    (if (in abbreviation e2umap)
      (.write fout (+ "<" meta-data ">\t" (get e2umap abbreviation)
                      "\t" a-literal " .\n")))))
```

Here is some of the generated RDF that has changed:

```
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>
  <http://dbpedia.org/page/IBM> .
<https://newsshop.com/may/a1023.html>
  <https://schema.org/Organization>
  <http://dbpedia.org/page/Banco_Esp%C3%ADrito_Santo> .
```

Now when we visualize generated RDF, we share nodes for The Wallstreet Journal and IBM:



Part of the RDF graph that shows shared nodes when URIs are used for RDF values instead of literal strings

While literal values sometimes are useful in generated RDF, using literals for the values in RDF triples prevents types of queries and inference that can be performed on the data.

Wrap-up

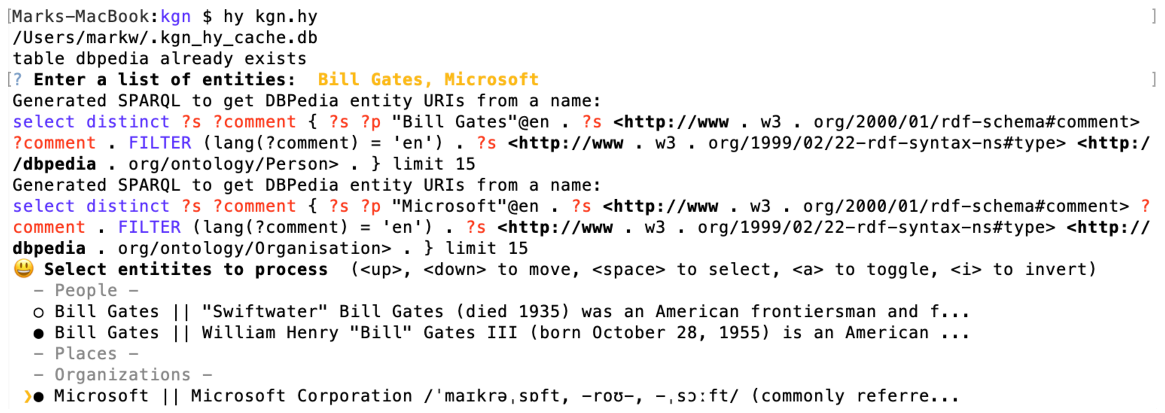
In the field of Artificial Intelligence there are two topics that get me the most excited and I have been fortunate to be paid to work on both: Deep Learning and Knowledge Graphs. Here we have just touched the surface for creating data for Knowledge Graphs but I hope that between this chapter and the material on RDF in the chapter **Datastores** that you have enough information and experience playing with the examples to get started prototyping a Knowledge Graph in your organizartion. My advice is to “start small” by picking a problem that your organization has that can be solved by not moving data around, but rather, by creating a custom Knowledge Graph for metadata for existing information in your organization.

Knowledge Graph Navigator

The Knowledge Graph Navigator (which I will often refer to as KGN) is a tool for processing a set of entity names and automatically exploring the public Knowledge Graph DBPedia[77] using SPARQL queries. I wrote KGN in Common Lisp for my own use to automate some things I used to do manually when exploring Knowledge Graphs, and later thought that KGN might be useful also for educational purposes. KGN uses NLP code developed in earlier chapters and we will reuse that code with a short review of using the APIs.

Please note that the example is a simplified version that I first wrote in Common Lisp and is also an example in my book Loving Common Lisp, or the Savvy Programmer's Secret Weapon[78] that you can read for free online. If you are interested you can see screen shots of the Common Lisp version here[79].

The following two screen shots show the text based user interface for this example. This example application asks the user for a list of entity names and uses SPARQL queries to discover potential matches in DBPedia. We use the python library PyInquirer[80] for requesting entity names and then to show the user a list of matches from DBPedia. The following screen shot shows these steps:



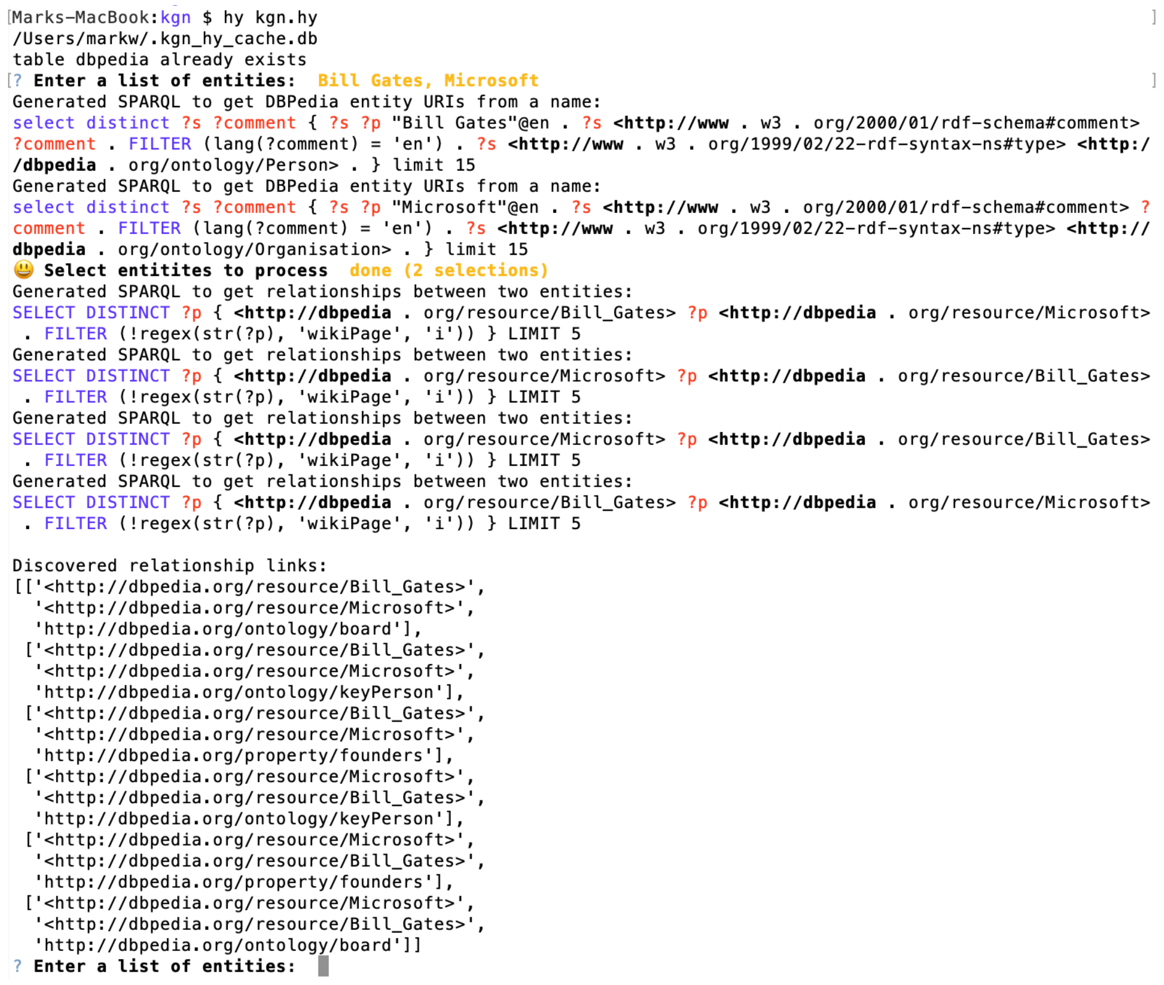
```
[Marks-MacBook:kgn $ hy kgn.hy                                                    ]
/Users/markw/.kgn_hy_cache.db
table dbpedia already exists
[? Enter a list of entities:  Bill Gates, Microsoft                               ]
Generated SPARQL to get DBPedia entity URIs from a name:
select distinct ?s ?comment { ?s ?p "Bill Gates"@en . ?s <http://www . w3 . org/2000/01/rdf-schema#comment>
?comment . FILTER (lang(?comment) = 'en') . ?s <http://www . w3 . org/1999/02/22-rdf-syntax-ns#type> <http:/
/dbpedia . org/ontology/Person> . } limit 15
Generated SPARQL to get DBPedia entity URIs from a name:
select distinct ?s ?comment { ?s ?p "Microsoft"@en . ?s <http://www . w3 . org/2000/01/rdf-schema#comment> ?
comment . FILTER (lang(?comment) = 'en') . ?s <http://www . w3 . org/1999/02/22-rdf-syntax-ns#type> <http://
dbpedia . org/ontology/Organisation> . } limit 15
😃 Select entitites to process  (<up>, <down> to move, <space> to select, <a> to toggle, <i> to invert)
  - People -
  ○ Bill Gates || "Swiftwater" Bill Gates (died 1935) was an American frontiersman and f...
  ● Bill Gates || William Henry "Bill" Gates III (born October 28, 1955) is an American ...
  - Places -
  - Organizations -
 ❯● Microsoft || Microsoft Corporation /ˈmaɪkrəˌsɒft, -roʊ-, -ˌsɔːft/ (commonly referre...
```

Initial user interaction with Knowledge Graph Navigator example

To select the entities of interest, the user uses a space character to select or deselect an entity and the return (or enter) key to accept the list selections.

After the user selects entities from the list, the list disappears. The next screen shot shows the output from this example after the user finishes selecting entities of interest:

[77] http://dbpedia.org
[78] https://leanpub.com/lovinglisp
[79] http://www.knowledgegraphnavigator.com/screen/
[80] https://github.com/CITGuru/PyInquirer



```
Marks-MacBook:kgn $ hy kgn.hy
/Users/markw/.kgn_hy_cache.db
table dbpedia already exists
? Enter a list of entities:  Bill Gates, Microsoft
Generated SPARQL to get DBPedia entity URIs from a name:
select distinct ?s ?comment { ?s ?p "Bill Gates"@en . ?s <http://www . w3 . org/2000/01/rdf-schema#comment>
?comment . FILTER (lang(?comment) = 'en') . ?s <http://www . w3 . org/1999/02/22-rdf-syntax-ns#type> <http:/
/dbpedia . org/ontology/Person> . } limit 15
Generated SPARQL to get DBPedia entity URIs from a name:
select distinct ?s ?comment { ?s ?p "Microsoft"@en . ?s <http://www . w3 . org/2000/01/rdf-schema#comment> ?
comment . FILTER (lang(?comment) = 'en') . ?s <http://www . w3 . org/1999/02/22-rdf-syntax-ns#type> <http://
dbpedia . org/ontology/Organisation> . } limit 15
😃 Select entitites to process  done (2 selections)
Generated SPARQL to get relationships between two entities:
SELECT DISTINCT ?p { <http://dbpedia . org/resource/Bill_Gates> ?p <http://dbpedia . org/resource/Microsoft>
 . FILTER (!regex(str(?p), 'wikiPage', 'i')) } LIMIT 5
Generated SPARQL to get relationships between two entities:
SELECT DISTINCT ?p { <http://dbpedia . org/resource/Microsoft> ?p <http://dbpedia . org/resource/Bill_Gates>
 . FILTER (!regex(str(?p), 'wikiPage', 'i')) } LIMIT 5
Generated SPARQL to get relationships between two entities:
SELECT DISTINCT ?p { <http://dbpedia . org/resource/Microsoft> ?p <http://dbpedia . org/resource/Bill_Gates>
 . FILTER (!regex(str(?p), 'wikiPage', 'i')) } LIMIT 5
Generated SPARQL to get relationships between two entities:
SELECT DISTINCT ?p { <http://dbpedia . org/resource/Bill_Gates> ?p <http://dbpedia . org/resource/Microsoft>
 . FILTER (!regex(str(?p), 'wikiPage', 'i')) } LIMIT 5

Discovered relationship links:
[['<http://dbpedia.org/resource/Bill_Gates>',
  '<http://dbpedia.org/resource/Microsoft>',
  'http://dbpedia.org/ontology/board'],
 ['<http://dbpedia.org/resource/Bill_Gates>',
  '<http://dbpedia.org/resource/Microsoft>',
  'http://dbpedia.org/ontology/keyPerson'],
 ['<http://dbpedia.org/resource/Bill_Gates>',
  '<http://dbpedia.org/resource/Microsoft>',
  'http://dbpedia.org/property/founders'],
 ['<http://dbpedia.org/resource/Microsoft>',
  '<http://dbpedia.org/resource/Bill_Gates>',
  'http://dbpedia.org/ontology/keyPerson'],
 ['<http://dbpedia.org/resource/Microsoft>',
  '<http://dbpedia.org/resource/Bill_Gates>',
  'http://dbpedia.org/property/founders'],
 ['<http://dbpedia.org/resource/Microsoft>',
  '<http://dbpedia.org/resource/Bill_Gates>',
  'http://dbpedia.org/ontology/board']]
? Enter a list of entities:
```

After the user selects entities of interest

The code for this application is in the directory **kgn**. You will need to install the following Python library that supports console/text user interfaces:

```
pip install PyInquirer
```

You will also need the **spacy** library and language model that we used in the earlier chapter on natural language processing. If you have not already done so, install these requirements:

```
pip install spacy
python -m spacy download en_core_web_sm
```

After listing the generated SPARQL for finding information for the entities in the query, KGN searches for relationships between these entities. These discovered relationships can be seen at the end of the last screen shot. Please note that this step makes SPARQL queries on **O(n^2)** where **n** is the number of entities. Local caching of SPARQL queries to DBPedia helps make processing many entities possible.

Every time KGN makes a SPARQL query web service call to DBPedia the query and response are cached in a SQLite database in ~/.**kgn_hy_cache.db** which can greatly speed up the program, especially in development mode when testing a set of queries. This caching also takes some load off of the public DBPedia endpoint, which is a polite thing to do.

Review of NLP Utilities Used in Application

We covered NLP in a previous chapter, so the following is just a quick review. The NLP code we use is near the top of the file **kgn.hy**:

```
(import spacy)

(setv nlp-model (spacy.load "en"))

(defn entities-in-text [s]
  (setv doc (nlp-model s))
  (setv ret {})
  (for
    [[ename etype] (lfor entity doc.ents [entity.text entity.label_])]
    (if (in etype ret)
        (setv (get ret etype) (+ (get ret etype) [ename]))
        (assoc ret etype [ename])))
  ret)
```

Here is an example use of this function:

```
=> (kgn.entities-in-text "Bill Gates, Microsoft, Seattle")
{'PERSON': ['Bill Gates'], 'ORG': ['Microsoft'], 'GPE': ['Seattle']}
```

The entity type “GPE” indicates that the entity is some type of location.

Developing Low-Level Caching SPARQL Utilities

While developing KGN and also using it as an end user, many SPARQL queries to DBPedia contain repeated entity names so it makes sense to write a caching layer. We use a SQLite database “~/.kgn_-hy_cache.db” to store queries and responses. We covered using SQLite in some detail in the chapter on datastores.

The caching layer is implemented in the file **cache.hy**:

```
(import [sqlite3 [connect version Error ]])
(import json)

(setv *db-path* "kgn_hy_cache.db")

(defn create-db []
  (try
    (setv conn (connect *db-path*))
    (print version)
    (setv cur (conn.cursor))
    (cur.execute "CREATE TABLE dbpedia (query string  PRIMARY KEY ASC, data json)")
    (conn.close)
    (except [e Exception] (print e))))

(defn save-query-results-dbpedia [query result]
  (try
    (setv conn (connect *db-path*))
    (setv cur (conn.cursor))
    (cur.execute "insert into dbpedia (query, data) values (?, ?)"
                 [query (json.dumps result)])
    (conn.commit)
    (conn.close)
    (except [e Exception] (print e))))

(defn fetch-result-dbpedia [query]
  (setv results [])
  (setv conn (connect *db-path*))
  (setv cur (conn.cursor))
  (cur.execute "select data from dbpedia where query = ? limit 1" [query])
  (setv d (cur.fetchall))
  (if (> (len d) 0)
      (setv results (json.loads (first (first d)))))
  (conn.close)
  results)

(create-db)
```

Here we store structured data from SPARQL queries as JSON data serialized as string values.

SPARQL Utilities

We will use the caching code from the last section and also the standard Python library **requests** to access the DBPedia servers. The following code is found in the file **sparql.hy** and also provides support for using both DBPedia and WikiData. We only use DBPedia in this chapter but when you start incorporating SPARQL queries into applications that you write, you will also probably want to use WikiData.

The function **do-query-helper** contains generic code for SPARQL queries and is used in functions **wikidata-sparql** and **dbpedia-sparql**:

```
(import json)
(import requests)
(require [hy.contrib.walk [let]])

(import [cache [fetch-result-dbpedia save-query-results-dbpedia]])

(setv wikidata-endpoint "https://query.wikidata.org/bigdata/namespace/wdq/sparql")
(setv dbpedia-endpoint "https://dbpedia.org/sparql")

(defn do-query-helper [endpoint query]
  ;; check cache:
  (setv cached-results (fetch-result-dbpedia query))
  (if (> (len cached-results) 0)
      (let ()
        (print "Using cached query results")
        (eval cached-results))
      (let ()
        ;; Construct a request
        (setv params { "query" query "format" "json"})

        ;; Call the API
        (setv response (requests.get endpoint :params params))

        (setv json-data (response.json))

        (setv vars (get (get json-data "head") "vars"))

        (setv results (get json-data "results"))

        (if (in "bindings" results)
            (let [bindings (get results "bindings")
                  qr
                  (lfor binding bindings
                        (lfor var vars
                              [var (get (get binding var) "value")]))]
              (save-query-results-dbpedia query qr)
              qr)
            []))))

(defn wikidata-sparql [query]
  (do-query-helper wikidata-endpoint query))

(defn dbpedia-sparql [query]
  (do-query-helper dbpedia-endpoint query))
```

Here is an example query (manually formatted for page width):

```
$ hy
hy 0.18.0 using CPython(default) 3.7.4 on Darwin
=> (import sparql)
table dbpedia already exists
=> (sparql.dbpedia-sparql
     "select ?s ?p ?o { ?s ?p ?o } limit 1")
[[['s', 'http://www.openlinksw.com/virtrdf-data-formats#default-iid'],
  ['p', 'http://www.w3.org/1999/02/22-rdf-syntax-ns#type'],
  ['o', 'http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat']]]
=>
```

This is a wild-card SPARQL query that will match any of the 9.5 billion RDF triples in DBPedia and return just one result.

This caching layer greatly speeds up my own personal use of KGN. Without caching, queries that contain many entity references simply take too long to run.

Utilities to Colorize SPARQL and Generated Output

When I first had the basic functionality of KGN working, I was disappointed by how the application looked as normal text. Every editor and IDE I use colorizes text in an appropriate way so I used standard ANSI terminal escape sequences to implement color hilting SPARQL queries.

The code in the following listing is in the file **colorize.hy**.

```
(require [hy.contrib.walk [let]])
(import [io [StringIO]])

;; Utilities to add ANSI terminal escape sequences to colorize text.
;; note: the following 5 functions return string values that then need to
;;       be printed.

(defn blue [s] (.format "{}{}{}" "\033[94m" s "\033[0m"))
(defn red [s] (.format "{}{}{}" "\033[91m" s "\033[0m"))
(defn green [s] (.format "{}{}{}" "\033[92m" s "\033[0m"))
(defn pink [s] (.format "{}{}{}" "\033[95m" s "\033[0m"))
(defn bold [s] (.format "{}{}{}" "\033[1m" s "\033[0m"))

(defn tokenize-keep-uris [s]
  (.split s))

(defn colorize-sparql [s]
  (let [tokens
        (tokenize-keep-uris
          (.replace (.replace (.replace s "{" " { ") "}" " } ") "." " . "))
        ret (StringIO)] ;; ret is an output stream for a string buffer
    (for [token tokens]
      (if (> (len token) 0)
          (if (= (get token 0) "?")
              (.write ret (red token))
              (if (in
                    token
                    ["where" "select" "distinct" "option" "filter"
                     "FILTER" "OPTION" "DISTINCT" "SELECT" "WHERE"])
                  (.write ret (blue token))
                  (if (= (get token 0) "<")
                      (.write ret (bold token))
                      (.write ret token)))))
      (if (not (= token "?"))
          (.write ret " ")))
    (.seek ret 0)
    (.read ret)))
```

You have seen colorized SPARQL in the two screen shots at the beginning of this chapter.

Text Utilities for Queries and Results

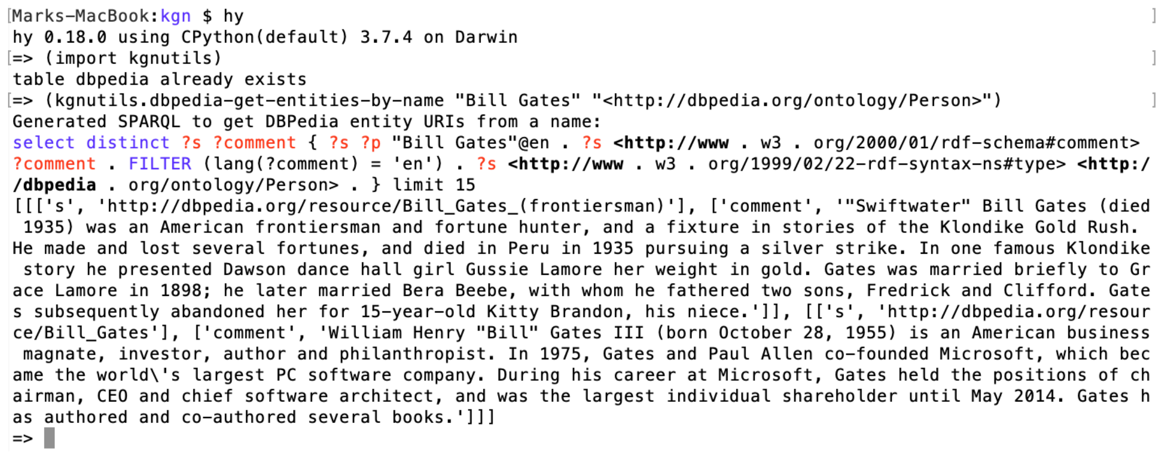
The application low level utility functions are in the file **kgn-utils.hy**. The function **dbpedia-get-entities-by-name** requires two arguments:

- The name of an entity to search for.
- A URI representing the entity type that we are looking for.

We embed a SPARQL query that has placeholders for the entity name and type. The filter expression specifies that we only want triple results with comment values in the English language by using **(lang(?comment) = 'en')**:

```
#!/usr/bin/env hy

(import [sparql [dbpedia-sparql]])
(import [colorize [colorize-sparql]])

(import [pprint [pprint]])
(require [hy.contrib.walk [let]])

(defn dbpedia-get-entities-by-name [name dbpedia-type]
  (let [sparql
        (.format "select distinct ?s ?comment {{ ?s ?p \"{}\"@en . ?s <http://www.w3\
.org/2000/01/rdf-schema#comment>  ?comment  . FILTER  (lang(?comment) = 'en') . ?s <\
http://www.w3.org/1999/02/22-rdf-syntax-ns#type> {} . }} limit 15" name dbpedia-type\
)]
    (print "Generated SPARQL to get DBPedia entity URIs from a name:")
    (print (colorize-sparql sparql))
    (dbpedia-sparql sparql)))
```

Here is an example:



```
Marks-MacBook:kgn $ hy
hy 0.18.0 using CPython(default) 3.7.4 on Darwin
=> (import kgnutils)
table dbpedia already exists
=> (kgnutils.dbpedia-get-entities-by-name "Bill Gates" "<http://dbpedia.org/ontology/Person>")
Generated SPARQL to get DBPedia entity URIs from a name:
select distinct ?s ?comment { ?s ?p "Bill Gates"@en . ?s <http://www . w3 . org/2000/01/rdf-schema#comment>
?comment . FILTER (lang(?comment) = 'en') . ?s <http://www . w3 . org/1999/02/22-rdf-syntax-ns#type> <http:/
/dbpedia . org/ontology/Person> . } limit 15
[[['s', 'http://dbpedia.org/resource/Bill_Gates_(frontiersman)'], ['comment', '"Swiftwater" Bill Gates (died
 1935) was an American frontiersman and fortune hunter, and a fixture in stories of the Klondike Gold Rush.
He made and lost several fortunes, and died in Peru in 1935 pursuing a silver strike. In one famous Klondike
 story he presented Dawson dance hall girl Gussie Lamore her weight in gold. Gates was married briefly to Gr
ace Lamore in 1898; he later married Bera Beebe, with whom he fathered two sons, Fredrick and Clifford. Gate
s subsequently abandoned her for 15-year-old Kitty Brandon, his niece.']], [['s', 'http://dbpedia.org/resour
ce/Bill_Gates'], ['comment', 'William Henry "Bill" Gates III (born October 28, 1955) is an American business
 magnate, investor, author and philanthropist. In 1975, Gates and Paul Allen co-founded Microsoft, which bec
ame the world\'s largest PC software company. During his career at Microsoft, Gates held the positions of ch
airman, CEO and chief software architect, and was the largest individual shareholder until May 2014. Gates h
as authored and co-authored several books.']]]
=>
```

Getting entities by name with colorized SPARL query script

Finishing the Main Function for KGN

We already looked at the NLP code near the beginning of the file **kgn.hy**. Let's look at the remainder of the implementation.

We need a dictionary (or hash table) to convert **spaCy** entity type names to DBPedia type URIs:

```
(setv entity-type-to-type-uri
      {"PERSON" "<http://dbpedia.org/ontology/Person>"
       "GPE" "<http://dbpedia.org/ontology/Place>"
       "ORG" "<http://dbpedia.org/ontology/Organisation>"
       })
```

When we get entity results from DBPedia, the comments describing entities can be a few paragraphs of text. We want to shorten the comments so they fit in a single line of the entity selection list that we have seen earlier. The following code defines a comment shortening function and also a global variable that we will use to store the entity URIs for each shortened comment:

```
1 (setv short-comment-to-uri {})
2
3 (defn shorten-comment [comment uri]
4   (setv sc (+ (cut comment 0 70) "..."))
5   (assoc short-comment-to-uri sc uri)
6   sc)
```

In line 5, we use the function **assoc** to add a key and value pair to an existing dictionary **short-comment-to-uri**.

Finally, let's look at the main application loop. In line 4 we are using the function **get-query** (defined in file **textui.hy**) to get a list of entity names from the user. In line 7 we use the function **entities-in-text** that we saw earlier to map text to entity types and names. In the nested loops in lines 13-26 we build one line descriptions of people, place, and organizations that we will use to show the user a menu for selecting entities found in DBPedia from the original query. We are giving the use a chance to select only the discovered entities that they are interested in.

In lines 33-35 we are converting the shortened comment strings the user selected back to DBPedia entity URIs. Finally in line 36 we use the function **entity-results->relationship-links** to find relationships between the user selected entities.

```
 1  (defn kgn []
 2    (while
 3      True
 4      (let [query (get-query)
 5            emap {}]
 6        (if (or (= query "quit") (= query "q"))
 7            (break))
 8        (setv elist (entities-in-text query))
 9        (setv people-found-on-dbpedia [])
10        (setv places-found-on-dbpedia [])
11        (setv organizations-found-on-dbpedia [])
12        (global short-comment-to-uri)
13        (setv short-comment-to-uri {})
14        (for [key elist]
15          (setv type-uri (get entity-type-to-type-uri key))
16          (for [name (get elist key)]
17            (setv dbp (dbpedia-get-entities-by-name name type-uri))
18            (for [d dbp]
19              (setv short-comment (shorten-comment (second (second d))
20              (second (first d))))
21              (if (= key "PERSON")
22                  (.extend people-found-on-dbpedia [(+ name  " || " short-comment)]))
23              (if (= key "GPE")
24                  (.extend places-found-on-dbpedia [(+ name  " || " short-comment)]))
25              (if (= key "ORG")
26                  (.extend organizations-found-on-dbpedia
27                  [(+ name  " || " short-comment)]))))))
28        (setv user-selected-entities
29              (select-entities
30                people-found-on-dbpedia
31                places-found-on-dbpedia
32                organizations-found-on-dbpedia))
```

```
      (setv uri-list [])
      (for [entity (get user-selected-entities "entities")]
        (setv short-comment (cut entity (+ 4 (.index entity " || "))))
        (.extend uri-list [(get short-comment-to-uri short-comment)]))
      (setv relation-data (entity-results->relationship-links uri-list))
      (print "\nDiscovered relationship links:")
      (pprint relation-data))))
```

If you have not already done so, I hope you experiment running this example application. The first time you specify an entity name expect some delay while DBPedia is accessed. Thereafter the cache will make the application more responsive when you use the same name again in a different query.

Wrap-up

If you enjoyed running and experimenting with this example and want to modify it for your own projects then I hope that I provided a sufficient road map for you to do so.

I got the idea for the KGN application because I was spending quite a bit of time manually setting up SPARQL queries for DBPedia and other public sources like WikiData, and I wanted to experiment with partially automating this exploration process.

Book Wrap-up

I love programming in Lisp languages but I often need to use Python libraries for Deep Learning and NLP. The Hy language is a good fit for me, it is simple to install along with the Python libraries that I use for my work and it is a fun language to write code in. Most importantly, Hy fits well with the type of iterative bottom-up REPL-based development that I prefer.

I hope that you enjoyed this short book and that at least a few things that you have learned here will both help you in your work and give you ideas for new personal projects.

Best regards,

Mark Watson

February 15, 2020