If you are running locally, you must download or create a database instance!!

Option 1: Download database instance

Run `download_data.sh` or execute the following commands manually at your terminal:

```
wget https://www.dropbox.com/s/2tbk02y8ezys4q9/snorkel.db.bz2
bunzip2 snorkel.db.bz2
```

Make certain `snorkel.db` is in the `tutorials/workshop/` directory.

Option 2: Create a new database instance

Run the Preprocessing Notebook Tutorial

Snorkel Workshop ¶

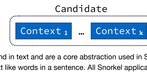
Part 1: Snorkel API

Complete Snorkel API documentation is available via Read the Docs

However, we provide several detailed examples below that are useful when you are using Snorkel for the first time.

In [ ]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline
from lib.init import *
from lib.util import check_exercise_1, check_exercise_2
```

I. Candidates and Contexts



`Candidate` objects represent potential mentions found in text and are a core abstraction used in Snorkel. `Candidate(s)` are defined over 1 or more `Context` objects, which are typically some unit of text like words in a sentence. All Snorkel applications require a custom Candidate class definition.

A. Example Definitions

In our tutorial, we define a `Spouse` relation as consisting of 2 `Span(s)` (i.e., sequences of words or characters) representing the mentions of 2 people that married. Defining a new `Candidate` class requires providing a name for the class (`Spouse`) and its `Span` arguments (`person1` and `person2`). The syntax for defining this relation is below:

In [ ]:
```
Spouse = candidate_subclass('Spouse', ['person1', 'person2'])
```

Alternatively, if we just want `Person` entities, we can define a `Candidate` that contains only 1 `Span` representing a person's name. Note how we only provide a list containing 1 argument now.

Exercise 1

In the cell below, define a new `Person` class containing on 1 `Span` argument with name `person`. Check your answer using the `check_exercise_1` function.

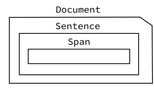
In [ ]:
```
# Define your Snorkel Person candidate type here


# check your type def (we use Spouse to show a wrong answer!)
check_exercise_1(Spouse)
```

B. Candidates in Context

By default, Snorkel candidates are defined over `Span` objects within a `Sentence` context. `Span(s)` correspond to conceptual categories in text like people or disease names. In the above example, our candidate represents the possible `Spouse` mention `(Barrack Obama, Michelle Obama)`. As readers, we know this mention is true due to external knowledge and the keyword of `wedding` occuring later in the sentence.

C. Context Hierarchy



All `Context(s)` are hierarchical in Snorkel. The default objects provided by Snorkel are show above.

II. Loading `Candidate(s)`

A. `Candidate` Member Functions and Variables

You will interact with candidates while writing labeling functions in Snorkel. The definition of the `Spouse` and `Span` classes is outlined below;

```
class Spouse(Candidate)
 Attributes:
    person1 (Span): relation argument
    person2 (Span): relation argument

class Span(Context)
 Methods:
    get_attrib_tokens(a="words"): return all tokens of the provided type a
    get_parent(): return parent Context
```

For the following examples, we'll look at the first candidate in our `cands` list. First we'll show the candidate in its parent sentence.

B. Querying Candiates from the Database

Once you've defined candidates as shown above, you need to do some preprocessing to load your documents, extract candidates, and then load everything into a database. This is a time consuming process, so we've pre-generated a database snapshot for you. Refer to our preprocessing tutorial Workshop_5 Advanced_Preprocessing for specific information on how this is done.

We assume that our Candidates have already been extracted and partitioned into `train`, `dev`, and `test` sets. For now, we will just load our `train` set candidates.

This query returns a list of candidate objects.

In [ ]:
```
cands = session.query(Candidate).filter(Candidate.split == 0).all()
```

Exercise 2

Find the candidate at index 222 in the `candidates` list initalized above. Check your answer using the `check_exercise_2` function.

In [ ]:
```
# check_exercise_2( YOUR CANDIDATE HERE )
```

III. Advanced Reference Materials

A. Accessing Parent `Context(s)`

Candidates live within Context objects. If we want to access the Context hierarchy, we can do so as follows:

In [ ]:
```
# we can access Span(s) as named member variables
print(cands[0].person1)
print(cands[0].person2)

# the raw word tokens for the person1 Span
print(cands[0].person1.get_attrib_tokens("words"))

# part of speech tags
print(cands[0].person1.get_attrib_tokens("pos_tags"))

# named entity recognition tags
print(cands[0].person1.get_attrib_tokens("ner_tags"))
```

In [ ]:
```
sentence = cands[0].get_parent()
document = sentence.get_parent()
```

B. Labeling Function Helpers

When writing labeling functions, there are several operators you will use over and over again; fetching text between span arguments, examing word windows around spans, etc.

Snorkel provides several core helper functions These are python helper functions that you can apply to candidates to return objects that are helpful during LF development.

You can (and should!) write your own helper functions to help write LFs.

In [ ]:
```
import re
from snorkel.lf_helpers import (
    get_left_tokens, get_right_tokens, get_between_tokens,
    get_text_between, get_tagged_text,
)
```

In [ ]:
```
print("Candidate LEFT tokens:   \t", list(get_left_tokens(cands[0],window=2)))
print("Candidate RIGHT tokens:  \t", list(get_right_tokens(cands[0],window=2)))
print("Candidate BETWEEN tokens:\t", get_text_between(cands[0]))
```

VI. Cheat Sheet

Jupyter notebooks provide a build in docstring display operator for functions. Just prepend `?` to any function name as shown below.

```
?get_left_tokens
```

For class member functions, don't forget to include the class name

```
?Span.get_attrib_tokens
```

Read the Docs

Complete Snorkel API documentation on Read the Docs

`Candidate` Helper Functions

Helper functions operate on a `Candidate` class instance, `c`.

```
get_left_tokens(c, window=3, attrib='words', n_max=1, case_sensitive=False)
get_right_tokens(c, window=3, attrib='words', n_max=1, case_sensitive=False)
get_between_tokens(c, attrib='words', n_max=1, case_sensitive=False)
get_text_between(c)
get_tagged_text(c)
```

A full list of helper functions is available at http://snorkel.readthedocs.io/en/master/etc.html#module-snorkel.lf_helpers

`Candidate` Member Functions

Give a `Candidate` class instance

```
.get_attrib_tokens(a='words')
.get_word_start()
.get_word_end()
```

`Sentence` Attributes

Variable Name	Description
`words`	Text Tokens
`lemmas`	Lemma, "a base word and its inflections"
`pos_tags`	Part-of-speech Tags
`ner_tags`	Named Entity Tags
`dep_parents`	Dependency Tree Heads
`dep_labels`	Dependency Tree Tags
`char_offsets`	Character Offsets
`abs_char_offsets`	Absolute (document) Character Offsets

Computing Labeling Function Metrics

`snorkel.lf_helpers.test_LF`

In [ ]:
```
```