FoLT Tutorial 4 Summary

PART 1: Span-Based Annotations

- A token or a sequence of tokens can be referred as a `span`:
  - We can define a `span` by slicing a `doc[start : end]`
    - a `doc` is usually a text that has been processed using a `pipeline` **(Chek Tutorial 2, Part 3)**
- Code that prints a span:

```
nlp = spacy.load("en_core_web_sm") #Load the pipeline
doc = nlp("Reviewers stated that limitations of available evidence mean \
that none of the prognostic models are at a stage where they could \
be used in clinical practice, so this question remains unanswered.") #pass the text
through the pipeline
span = doc[3:26] #slice it
print(span)
```

- Result:

```
limitations of available evidence mean that none of the prognostic models are at a stage where they could be used in clinical practice
```

- Now we can operate on the span :
  - get the number of tokens **(I)**
  - get a token using its index **(II)**
  - iterate over the tokens and use their lexical attributes **(Check Tutorial 3, Part 4) (III)**
  - we can can also get the `lemma_` of the whole span/split **(IV)**
  - we can retrieve tokens that satisfy a certain POS, eg. noun, verb... **(V)**
- Code to demonstrate what we talked about:

```
print(len(span)) # I
print(span[0]) # II
print([(t.text, t.pos_, t.dep_, t.lemma_) for t in span]) # III
print(span.lemma_)                              #--| IV
print(" ".join([t.lemma_ for t in span])) #--| IV
print(list(span.noun_chunks)) # V
```

- Result:

```
23
limitations
[('limitations', 'NOUN', 'nsubj', 'limitation'), ('of', 'ADP', 'prep', 'of'), ('available', 'ADJ', 'amod', 'available'), ('evidence',
limitation of available evidence mean that none of the prognostic model be at a stage where they could be use in clinical practice
limitation of available evidence mean that none of the prognostic model be at a stage where they could be use in clinical practice
[limitations, available evidence, none, the prognostic models, a stage, they, clinical practice]
```

- Things you should know:
  - you can set and get a span's `label`
  - you have to use `label_` for that, or you'll get the hash value of the label
- Code:

```
span.label_ = "negative"
print(span.label_)
print(span.label)
```

- Result:

```
negative
11803922482410560765
```

PART 2: Inter-annotator Agreement

- Remember Cohen's Cappa from Lecture 1, well here is a reminder if you don't
    - **Cohen's Kappa**: $\kappa = \frac{p_o - p_e}{1 - p_e}$, measuring how 2 annotaters agree with each other above chance.
    - Now that you remember we will make a function that calculate it, how it works:
        1. we calculate the p_o **(observed agreement)**
        2. we calcualte the p_e **(expected agreement)**
        3. then we use the κ formula
- Code:

```
def cohen_kappa (annotation_1, annotation_2, labels):
    po = len([[id, label1, label2] for [id, label1], [id, label2] in zip(annotation_1, 
annotation_2) if label1 == label2]) / len(annotation_1)
    pe = 0
    for l in labels:
        p1 = len([[id, label] for [id, label] in annotation_1 if label == l ]) / 
len(annotation_1)
        p2 = len([[id, label] for [id, label] in annotation_2 if label == l ]) / 
len(annotation_2)
        pe = pe + p1 * p2
    k = (po - pe) / (1 - pe)
    return k
```

- Example:

```
annotation_1 = [["0", "positive"], ["1", "negative"], ["2", "negative"], ["3", 
"positive"], ["4", "neutral"], ["5", "positive"]]
annotation_2 = [["0", "positive"], ["1", "negative"], ["2", "positive"], ["3", 
"neutral"], ["4", "positive"], ["5", "positive"]]
labels = ["positive", "negative", "neutral"]
cohen_kappa (annotation_1, annotation_2, labels)
```

- Result

```
0.1428571428571429
```

- Review: bad agreement, $\kappa < 20$ is poor agreement

PART 3: Other Inter-annotator Agreement Metrics

- **Limitations of Cohen's Cappa:** only used to measure agreement between 2 annotators and it's categorical
- Other Methods:
    - **Fleiss'kappa:** $\kappa = \frac{p_o - p_e}{1 - p_e}$, is an extension of Cohen's for `three annotators or more` , to a `fixed number of items` , at the `condition` that `for each item annotators are randomly sampled` . It's designed for situations where a `fixed number of annotators` assess `a set of items` and `different items can be rated by different annotators` . E.g Item 1 is annotated by annotator A, B, and C; but Item 2 could be annotated by annotator D, E, and F.
    - **Krippendorff's Alpha**: $\alpha = \frac{p_a - p_e}{1 - p_e}$, based on calculating percentage agreement, it can:
        - Calculate IAA for incomplete data

- Compare an arbitary number of annotators
- Handle “shades of gray” where annotators might only partially agree with each other