Quebap is a library which is designed to generalize dataset and model structure so that if one adds a new model one can test it on a range of NLP and NLU tasks, and vice versa, if one had a new dataset one can try all the models available in Quebap. Thus with Quebap we hope to push the breadth and depth of research: You design a new model and other researchers and others are immediately able to compare against it. If you design a new dataset you immediately have a range of baseline from the models used in Quebap. Other profit from your efforts and others profit from yours.

Quebap is best understood by going from the high-level function and classes to low-level function and classes. The highest level entry point of Quebap is the training\_pipeline.py. The training pipeline calls other high-level functions step-by-step, from input data, over preprocessing and data wrangling, to training the selected model. Although the training\_pipeline.py script is more complex, here are some other high level functions and classes which are traversed while going along the pipeline from data to model.

The script does step-by-step:

1. Define sisyphos models

* Predefined models found in quebap.sisphos.models such as:
  + Conditional reader: Two bidirectional LSTM over two sequences where the second is conditioned on the final state of the other
  + Attentive reader : Like conditional reader, but all states are processed with attention so that the most important bits of each of the two sequences are filtered from the unimportant bits. Finally these two streams of filtered information are combined

1. Parse the input arguments

* Standard argparse. Arguments include:
  + Batchsize, pretrain embeddings, learning rate, l2 penalty, clip value (gradients), dropout, epochs, negative sampling

1. Read the train, dev, and test data

* Uses quebap.load.read\_quebap.quebap\_load which loads a JSON file in a specific format. To transform these data into this JSON format there exist wrangling files for certain data sets under the path “quebap/quebap/load/”. The JSON format can be seen as a python dictionary which high level names for different kind of data:
  + Question (Q): Question text or binary relation like (Donald Trump, presidentOf, X)
  + Support (S): Supportive text passage for the question.
  + Candidates (C): A corpus may have 10000 entities, but for this question only 10 candidates are relevant (2 are correct), for example all entities in the supporting passage are candidates. Candidates might also refer to all words in the vocabulary (no restrictions).
  + Answers: The answers to the question or binary relations. This may also a span like (17, 83) indicating the answer is located between character position 17 and 83 (Stanford SQuAD)
* At this point in the pipeline one can also load pretrained embeddings

1. Preprocesses the data (tokenize, normalize, add start and end of sentence tags) via the sisyphos.pipeline method

* This is the heaviest and most detailed processing step. In the script this data wrangling and preprocessing pipeline is called with a simple quebap.sisyphos.pipelines.pipeline(..) but behind this method there are several preprocessing steps:
  + Quebap.sisyphs.map.deep\_map: This is a clever method which traverses a dictionary for certain keys and transforms the values of given keys in-place in a very efficient manner. It does this by using a map function to the list of value under the given dictionary keys. It is usually used to transform a list of question strings, into a tokenized version, that is transform it into a list of question word-lists
  + Quebap.sisyphs.map.deep\_seq\_map: The sister of deep\_map. Also applies a function and transforms the given values under a dictionary keys in-place. The difference is that it applies this functionality on lists of lists (for example tokenized questions). With that we can use this function to do many things:
    - Words -> lower case words
    - Words -> ids (and then use these ids for indices of word embeddings; this is done with the Vocab class below)
    - Words -> Pad words with beginning and end of sentence tag, that is

[Word1, word2, word3] -> [SOS, word1, word2, word3, EOS] would be done with deep map in this way:

deep\_seq\_map(corpus, lambda xs: ["<SOS>"] + xs + ["<EOS>"], ['question'])

* + Class Quebap.sisyphos.vocab.Vocab: This class builds a vocabulary from tokens (usually words) assigns an identifier to each word and maintains this map from id to word and from word to id. This class also works together with pretrained vocabularies which are then extended through more data

1. Create NeuralVocab

* A word to embedding class which manages the training of embedding which optionally may be enriched with some pretrained embeddings. Parameters may be frozen and there are options for a projection layer to reduce the size of the inputs into the next layer and to normalize embeddings to unit norm.

(6) Create TensorFlow placeholders and initialize model

(7) Batch the data via sisyphos.batch.get\_feed\_dicts

(8) Add hooks

* Hooks are functions which are invoked after either the end of an iteration or the end of an epoch. They usually print some information (loss value, time taken this epoch, ETA until the model is fully trained, statistics of certain tensors like weights) and save this information to the TensorFlow summary writer so that these data can be visualized via TensorBoard.

(9) Train the model

* Calls quebap.sisyphos.train.train with everything which was constructed in the previous steps such as the batched data, the model, the hooks, and the parameters