TirMorph Quick How To

Na-Rae Han, naraehan@pitt.edu, August 2017

In short, TirMorph is a morphological parser for the Tigrinya language, written on the `morpar` library.
The script is called `tir_morph.py` (was renamed). On Lor, it is found in `/usr2/data/shared/LoReHLT17/internal/Morph/Tir/v5/`.
That's where you will also find a copy of this How-to document.

There are four user-facing functions. Start by importing them. Additionally, there are some functions and sample texts you can import for testing purposes.

```
>>> import tir_morph
>>> from tir_morph import fullparse, best_fullparse, parse, best_parse
>>> from tir_morph import jsonprint, g2pp, sample, sample_text
```

The function `fullparse()` returns the full parsing output for a given token. To try out with the first sample word meaning 'houses':

```
>>> word = sample[0]              # Let's take the first sample word.
>>> print(word)
ገዛውቲ
>>> print(g2pp(word))             # g2pp converts Ge'ez to Epitran IPA (tir-Ethi-pp mode.)
gəzawti
>>> fullparse(word)               # Bunch of unicode strings. Hurts eyes.
[{u'breakdown': u'\u0261\u0259za-w\u0268ti', u'definition': u'home', u'natural': u'multiple home',
u'gloss': u'home-PL', u'lemma': u'\u0261\u0259za', u'cost': u'XXXXXXX'}, {u'breakdown':
u'\u0261\u0259za-w\u0268ti', u'definition': u'house', u'natural': u'multiple house', u'gloss':
u'house-PL', u'lemma': u'\u0261\u0259za', u'cost': u'XXXXXXX'}, {u'breakdown': u'\u0261\u0259za-w\u0268ti',
u'definition': u'building', u'natural': u'multiple building', u'gloss': u'building-PL', u'lemma':
u'\u0261\u0259za', u'cost': u'XXXXXXXX'}]
>>> jsonprint(fullparse(word))            # Easier to read!
[{"breakdown": "gəza-wɨti", "definition": "home", "natural": "multiple home", "gloss": "home-PL",
"lemma": "gəza", "cost": "XXXXXXX"}, {"breakdown": "gəza-wɨti", "definition": "house", "natural":
"multiple house", "gloss": "house-PL", "lemma": "gəza", "cost": "XXXXXXX"}, {"breakdown": "gəza-wɨti",
"definition": "building", "natural": "multiple building", "gloss": "building-PL", "lemma": "gəza", "cost":
"XXXXXXXX"}]
>>>
```

By default, `fullparse()` returns three top parses as a list, which are indicated by the cost strings `"xxxxxxxxxxxxx"`. Usually this will be sufficient, but if not you can have more flexibility by specifying `top=0` or other parameters. See the docstring for more information, accessible via `help(fullparse)`.

If you want the top parse only, use `best_fullparse()` instead, which returns the first (i.e., top-ranked) dictionary object from the list:

```
>>> jsonprint(best_fullparse(word)
{"breakdown": "gəza-wɨti", "definition": "home", "natural": "multiple home", "gloss": "home-PL",
"lemma": "gəza", "cost": "XXXXXXX"}
```

So that gives you the full picture of the morphological parse across multiple **channels**. Details:

- **lemma** -- Stem after removing all affixes, e.g., *ʔertɨrawjan* --> *ʔertɨra*
- **gloss** -- Stem's meaning plus grammatical information provided by affixes

    - *be-3SG.MASC* ---> 3rd person singular masculine form of 'be'
    - *təħħiza-PL* ---> plural form of a guessed stem *təħħiza*
- **breakdown** -- Full form with morpheme boundaries indicated by '-'
    - *ʔertɨra-wɨjan*, *mɨ-t͡sʼɨħɨf-ɨti*
- **definition** -- Lemma's meaning (in English) pulled from a dictionary.
    - Empty string "" if stem is guessed.
- **natural** -- Natural-sounding English reading of the word.
- **cost** -- Cost of the parse in string length.
    - XXXXX indicates a very small cost (item could be directly off of a dictionary entry) which means high confidence.
    - XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX indicates low confidence and a guessed stem with no dictionary hit.

Since everyone is working on different NLP tasks, you might want to focus on a particular channel output. `parse(w, channel)` and `best_parse(w, channel)` let you query a single channel.

```
>>> jsonprint(parse(word, 'gloss'))
["home-PL", "house-PL", "building-PL"]        # Top 3 glosses returned as a list. PL means 'plural'.
>>> jsonprint(best_parse(word, 'gloss'))
"home-PL"                                     # Top gloss returned as a string
```

Let's try this on a piece of text:

```
>>> print(sample_text)
ካብቲ ንበልዖ መግብን ንሰትዮ ማይን ጀሚርካ ፡ ኣብ ማእሰርቲ ንሰብ ዘይግባእ ሕሱም ኣተሓሕዛ እዩ ዘለዎም ።
>>> for w in sample_text.split():
        jsonprint(parse(w, 'lemma'))

["kab", "kab", "kab"]      # These are real Tigrinya words from a dictionary.
["bəlʕo", "nɨbəlʕo"]       # These are guessed stems. See below.
["məgəbə"]
["sət", "sət", "nɨsət"]    # These are also guessed. See below.
["maj", "maj", "majɨn"]
["d͡ʒəmərə"]
[":"]
["ʔab", "ʔab", "ʔab"]
["maʔsərti", "maʔsərti", "maʔsərti"]
["səb", "səb", "səb"]
["zəjgɨbaʔ", "zəjgɨbaʔ", "zəjgɨbaʔ"]
["ħɨsum", "ħɨsum", "ħɨsum"]
["ʔatəhaħza", "təħħiza", "ʔɨtəħħiza"]
["ʔɨju", "ʔɨju", "ʔɨju"]
["ʔalo"]
["።"]
>>> for w in sample_text.split():
        jsonprint(parse(w, 'definition'))

["of", "from", "than"]
["", ""]                 # Empty definitions because stems were guessed.
["feed"]
["", "", ""]             # Likewise, all guessed stems.
["water", "water", ""]
["go"]
[""]
["in", "on", "at"]
["prison", "bondage", "imprisonment"]
["man", "being", "body"]
```

```
["inappropriate", "unsuitable", "unwise"]
["mean", "bad", "evil"]
["treatment", "", ""]
["is", "be", "be (descriptive)"]
["exist"]
[""]
```

Note the relationship between the **lemma** channel and the **definition** channel. It's important to recognize that not all lemmas are legal Tigrinya words/stems -- some of them are best guesses. Consulting the definition channel reveals whether or not the lemmas are real Tigrinya words. Another method is by consulting the **cost** channel:

```
>>> for w in sample_text.split():
        jsonprint(parse(w, 'cost'))

["XXX", "XXXXX", "XXXXXX"]      # <- Successful hit in a dictionary.
["XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
["XXXXXX"]                           # ^^^ These are not.
["XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", "
["XXXXXXXXXXXXXXXXX", "XXXXXXXXXXXXXXXXXXX", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"]
["XXXXXX"]
[""]
["XX", "XXX", "XXX"]
["XXXXXXXX", "XXXXXXXXXX", "XXXXXXXXXX"]
["XXXXXX", "XXXXXXX", "XXXXXXXX"]
["XXXXXXXXXX", "XXXXXXXXXX", "XXXXXXXXXX"]
["XXXXXX", "XXXXXXX", "XXXXXXX"]
["XXXXXXX", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX", "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
["XX", "XXX", "XXXXXXXXXXXXXXXXXX"]
["XXXXXX"]
[""]
```

You can see that the guessed lemmas come with distinctively high costs.

So, if you find yourself cross-referencing between multiple channels, you might want to consider obtaining `fullparse()` or `best_fullparse()` instead for all channels and then lookup the dictionary object:

```
>>> housebest = best_fullparse(word)   # a dictionary
>>> jsonprint(housebest['lemma'])
"gəza"
>>> jsonprint(housebest['gloss'])      # 'gloss' as a key
"home-PL"
>>> jsonprint(housebest['cost'])
"XXXXXXX"
>>>
```

Questions? Comments? Find me on the slack channel.