WordNet Interface

WordNet is just another NLTK corpus reader, and can be imported like this:

```
>>> from nltk.corpus import wordnet
```

For more compact code, we recommend:

```
>>> from nltk.corpus import wordnet as wn
```

Words

Look up a word using `synsets()`; this function has an optional `pos` argument which lets you constrain the part of speech of the word:

```
>>> wn.synsets('dog') # doctest: +ELLIPSIS +NORMALIZE_WHITESPACE
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'), Synset('chase.v.01')]
>>> wn.synsets('dog', pos=wn.VERB)
[Synset('chase.v.01')]
```

The other parts of speech are `NOUN`, `ADJ` and `ADV`. A synset is identified with a 3-part name of the form: word.pos.nn:

```
>>> wn.synset('dog.n.01')
Synset('dog.n.01')
>>> print(wn.synset('dog.n.01').definition())
a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric times;
>>> len(wn.synset('dog.n.01').examples())
1
>>> print(wn.synset('dog.n.01').examples()[0])
the dog barked all night
>>> wn.synset('dog.n.01').lemmas()
[Lemma('dog.n.01.dog'), Lemma('dog.n.01.domestic_dog'), Lemma('dog.n.01.Canis_familiaris')]
>>> [str(lemma.name()) for lemma in wn.synset('dog.n.01').lemmas()]
['dog', 'domestic_dog', 'Canis_familiaris']
>>> wn.lemma('dog.n.01.dog').synset()
Synset('dog.n.01')
```

The WordNet corpus reader gives access to the Open Multilingual WordNet, using ISO-639 language codes.

```
>>> sorted(wn.langs())
['als', 'arb', 'cat', 'cmn', 'dan', 'eng', 'eus', 'fas',
'fin', 'fra', 'fre', 'glg', 'heb', 'ind', 'ita', 'jpn', 'nno',
'nob', 'pol', 'por', 'spa', 'tha', 'zsm']
>>> wn.synsets(b'\xe7\x8a\xac'.decode('utf-8'), lang='jpn')
[Synset('dog.n.01'), Synset('spy.n.01')]
>>> wn.synset('spy.n.01').lemma_names('jpn')
['\u3044\u306c', '\u307e\u308f\u3057\u8005', '\u30b9\u30d1\u30a4', '\u56de\u3057\u8005',
'\u56de\u8005', '\u5bc6\u5075', '\u5de5\u4f5c\u54e1', '\u5efb\u3057\u8005',
'\u5efb\u8005', '\u63a2', '\u63a2\u308a', '\u72ac', '\u79d8\u5bc6\u635c\u67fb\u54e1',
'\u8adc\u5831\u54e1', '\u8adc\u8005', '\u9593\u8005', '\u9593\u8adc', '\u96a0\u5bc6']
>>> wn.synset('dog.n.01').lemma_names('ita')
['cane', 'Canis_familiaris']
>>> wn.lemmas('cane', lang='ita')
[Lemma('dog.n.01.cane'), Lemma('hammer.n.01.cane'), Lemma('cramp.n.02.cane'),
Lemma('bad_person.n.01.cane'), Lemma('incompetent.n.01.cane')]
>>> sorted(wn.synset('dog.n.01').lemmas('dan'))
[Lemma('dog.n.01.hund'), Lemma('dog.n.01.k\xf8ter'),
Lemma('dog.n.01.vovhund'), Lemma('dog.n.01.vovse')]
>>> sorted(wn.synset('dog.n.01').lemmas('por'))
[Lemma('dog.n.01.cachorro'), Lemma('dog.n.01.c\xe3es'),
Lemma('dog.n.01.c\xe3o'), Lemma('dog.n.01.c\xe3o')]
>>> dog_lemma = wn.lemma(b'dog.n.01.c\xc3\xa3o'.decode('utf-8'), lang='por')
>>> dog_lemma
Lemma('dog.n.01.c\xe3o')
>>> dog_lemma.lang()
'por'
>>> len(wordnet.all_lemma_names(pos='n', lang='jpn'))
66027
```

Synsets

Synset: a set of synonyms that share a common meaning.

```
>>> dog = wn.synset('dog.n.01')
>>> dog.hypernyms()
[Synset('canine.n.02'), Synset('domestic_animal.n.01')]
>>> dog.hyponyms()  # doctest: +ELLIPSIS
[Synset('basenji.n.01'), Synset('corgi.n.01'), Synset('cur.n.01'), Synset('dalmatian.n.02'), ...]
>>> dog.member_holonyms()
[Synset('canis.n.01'), Synset('pack.n.06')]
>>> dog.root_hypernyms()
[Synset('entity.n.01')]
>>> wn.synset('dog.n.01').lowest_common_hypernyms(wn.synset('cat.n.01'))
[Synset('carnivore.n.01')]
```

Each synset contains one or more lemmas, which represent a specific sense of a specific word.

Note that some relations are defined by WordNet only over Lemmas:

```
>>> good = wn.synset('good.a.01')
>>> good.antonyms()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Synset' object has no attribute 'antonyms'
```

```
>>> good.lemmas()[0].antonyms()
[Lemma('bad.a.01.bad')]
```

The relations that are currently defined in this way are *antonyms*, *derivationally_related_forms* and *pertainyms*.

Lemmas

```
>>> eat = wn.lemma('eat.v.03.eat')
>>> eat
Lemma('feed.v.06.eat')
>>> print(eat.key())
eat%2:34:02::
>>> eat.count()
4
>>> wn.lemma_from_key(eat.key())
Lemma('feed.v.06.eat')
>>> wn.lemma_from_key(eat.key()).synset()
Synset('feed.v.06')
>>> wn.lemma_from_key('feebleminded%5:00:00:retarded:00')
Lemma('backward.s.03.feebleminded')
>>> for lemma in wn.synset('eat.v.03').lemmas():
...     print(lemma, lemma.count())
...
Lemma('feed.v.06.feed') 3
Lemma('feed.v.06.eat') 4
>>> for lemma in wn.lemmas('eat', 'v'):
...     print(lemma, lemma.count())
...
Lemma('eat.v.01.eat') 61
Lemma('eat.v.02.eat') 13
Lemma('feed.v.06.eat') 4
Lemma('eat.v.04.eat') 0
Lemma('consume.v.05.eat') 0
Lemma('corrode.v.01.eat') 0
```

Lemmas can also have relations between them:

```
>>> vocal = wn.lemma('vocal.a.01.vocal')
>>> vocal.derivationally_related_forms()
[Lemma('vocalize.v.02.vocalize')]
>>> vocal.pertainyms()
[Lemma('voice.n.02.voice')]
>>> vocal.antonyms()
[Lemma('instrumental.a.01.instrumental')]
```

The three relations above exist only on lemmas, not on synsets.

Verb Frames

```
>>> wn.synset('think.v.01').frame_ids()
[5, 9]
>>> for lemma in wn.synset('think.v.01').lemmas():
...     print(lemma, lemma.frame_ids())
...     print(" | ".join(lemma.frame_strings()))
...
Lemma('think.v.01.think') [5, 9]
Something think something Adjective/Noun | Somebody think somebody
Lemma('think.v.01.believe') [5, 9]
Something believe something Adjective/Noun | Somebody believe somebody
Lemma('think.v.01.consider') [5, 9]
Something consider something Adjective/Noun | Somebody consider somebody
Lemma('think.v.01.conceive') [5, 9]
Something conceive something Adjective/Noun | Somebody conceive somebody
>>> wn.synset('stretch.v.02').frame_ids()
[8]
>>> for lemma in wn.synset('stretch.v.02').lemmas():
...     print(lemma, lemma.frame_ids())
...     print(" | ".join(lemma.frame_strings()))
...
Lemma('stretch.v.02.stretch') [8, 2]
Somebody stretch something | Somebody stretch
Lemma('stretch.v.02.extend') [8]
Somebody extend something
```

Similarity

```
>>> dog = wn.synset('dog.n.01')
>>> cat = wn.synset('cat.n.01')

>>> hit = wn.synset('hit.v.01')
>>> slap = wn.synset('slap.v.01')
```

`synset1.path_similarity(synset2):` Return a score denoting how similar two word senses are, based on the shortest path that connects the senses in the is-a (hypernym/hypnoym) taxonomy. The score is in the range 0 to 1. By default, there is now a fake root node added to verbs so for cases where previously a path could not be found---and None was returned---it should return a value. The old behavior can be achieved by setting simulate_root to be False. A score of 1 represents identity i.e. comparing a sense with itself will return 1.

```
>>> dog.path_similarity(cat)  # doctest: +ELLIPSIS
0.2...

>>> hit.path_similarity(slap)  # doctest: +ELLIPSIS
0.142...

>>> wn.path_similarity(hit, slap)  # doctest: +ELLIPSIS
0.142...
```

```
>>> print(hit.path_similarity(slap, simulate_root=False))
None

>>> print(wn.path_similarity(hit, slap, simulate_root=False))
None
```

`synset1.lch_similarity(synset2)`: Leacock-Chodorow Similarity: Return a score denoting how similar two word senses are, based on the shortest path that connects the senses (as above) and the maximum depth of the taxonomy in which the senses occur. The relationship is given as -log(p/2d) where p is the shortest path length and d the taxonomy depth.

```
>>> dog.lch_similarity(cat)  # doctest: +ELLIPSIS
2.028...

>>> hit.lch_similarity(slap)  # doctest: +ELLIPSIS
1.312...

>>> wn.lch_similarity(hit, slap)  # doctest: +ELLIPSIS
1.312...

>>> print(hit.lch_similarity(slap, simulate_root=False))
None

>>> print(wn.lch_similarity(hit, slap, simulate_root=False))
None
```

`synset1.wup_similarity(synset2)`: Wu-Palmer Similarity: Return a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node). Note that at this time the scores given do _not_ always agree with those given by Pedersen's Perl implementation of Wordnet Similarity.

The LCS does not necessarily feature in the shortest path connecting the two senses, as it is by definition the common ancestor deepest in the taxonomy, not closest to the two senses. Typically, however, it will so feature. Where multiple candidates for the LCS exist, that whose shortest path to the root node is the longest will be selected. Where the LCS has multiple paths to the root, the longer path is used for the purposes of the calculation.

```
>>> dog.wup_similarity(cat)  # doctest: +ELLIPSIS
0.857...

>>> hit.wup_similarity(slap)
0.25

>>> wn.wup_similarity(hit, slap)
0.25

>>> print(hit.wup_similarity(slap, simulate_root=False))
None

>>> print(wn.wup_similarity(hit, slap, simulate_root=False))
None
```

`wordnet_ic` Information Content: Load an information content file from the wordnet_ic corpus.

```
>>> from nltk.corpus import wordnet_ic
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
>>> semcor_ic = wordnet_ic.ic('ic-semcor.dat')
```

Or you can create an information content dictionary from a corpus (or anything that has a words() method).

```
>>> from nltk.corpus import genesis
>>> genesis_ic = wn.ic(genesis, False, 0.0)
```

`synset1.res_similarity(synset2, ic)`: Resnik Similarity: Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node). Note that for any similarity measure that uses information content, the result is dependent on the corpus used to generate the information content and the specifics of how the information content was created.

```
>>> dog.res_similarity(cat, brown_ic)  # doctest: +ELLIPSIS
7.911...
>>> dog.res_similarity(cat, genesis_ic)  # doctest: +ELLIPSIS
7.204...
```

`synset1.jcn_similarity(synset2, ic)`: Jiang-Conrath Similarity Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The relationship is given by the equation 1 / (IC(s1) + IC(s2) - 2 * IC(lcs)).

```
>>> dog.jcn_similarity(cat, brown_ic)  # doctest: +ELLIPSIS
0.449...
>>> dog.jcn_similarity(cat, genesis_ic)  # doctest: +ELLIPSIS
0.285...
```

`synset1.lin_similarity(synset2, ic)`: Lin Similarity: Return a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. The relationship is given by the equation 2 * IC(lcs) / (IC(s1) + IC(s2)).

```
>>> dog.lin_similarity(cat, semcor_ic)  # doctest: +ELLIPSIS
0.886...
```

Access to all Synsets

Iterate over all the noun synsets:

```
>>> for synset in list(wn.all_synsets('n'))[:10]:
...     print(synset)
...
Synset('entity.n.01')
Synset('physical_entity.n.01')
Synset('abstraction.n.06')
Synset('thing.n.12')
Synset('object.n.01')
Synset('whole.n.02')
Synset('congener.n.03')
Synset('living_thing.n.01')
```

```
Synset('organism.n.01')
Synset('benthos.n.02')
```

Get all synsets for this word, possibly restricted by POS:

```
>>> wn.synsets('dog') # doctest: +ELLIPSIS
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'), ...]
>>> wn.synsets('dog', pos='v')
[Synset('chase.v.01')]
```

Walk through the noun synsets looking at their hypernyms:

```
>>> from itertools import islice
>>> for synset in islice(wn.all_synsets('n'), 5):
...     print(synset, synset.hypernyms())
...
Synset('entity.n.01') []
Synset('physical_entity.n.01') [Synset('entity.n.01')]
Synset('abstraction.n.06') [Synset('entity.n.01')]
Synset('thing.n.12') [Synset('physical_entity.n.01')]
Synset('object.n.01') [Synset('physical_entity.n.01')]
```

Morphy

Look up forms not in WordNet, with the help of Morphy:

```
>>> wn.morphy('denied', wn.NOUN)
>>> print(wn.morphy('denied', wn.VERB))
deny
>>> wn.synsets('denied', wn.NOUN)
[]
>>> wn.synsets('denied', wn.VERB) # doctest: +NORMALIZE_WHITESPACE
[Synset('deny.v.01'), Synset('deny.v.02'), Synset('deny.v.03'), Synset('deny.v.04'),
Synset('deny.v.05'), Synset('traverse.v.03'), Synset('deny.v.07')]
```

Morphy uses a combination of inflectional ending rules and exception lists to handle a variety of different possibilities:

```
>>> print(wn.morphy('dogs'))
dog
>>> print(wn.morphy('churches'))
church
>>> print(wn.morphy('aardwolves'))
aardwolf
>>> print(wn.morphy('abaci'))
abacus
>>> print(wn.morphy('book', wn.NOUN))
book
>>> wn.morphy('hardrock', wn.ADV)
>>> wn.morphy('book', wn.ADJ)
>>> wn.morphy('his', wn.NOUN)
>>>
```

Synset Closures

Compute transitive closures of synsets

```
>>> dog = wn.synset('dog.n.01')
>>> hypo = lambda s: s.hyponyms()
>>> hyper = lambda s: s.hypernyms()
>>> list(dog.closure(hypo, depth=1)) == dog.hyponyms()
True
>>> list(dog.closure(hyper, depth=1)) == dog.hypernyms()
True
>>> list(dog.closure(hypo))
[Synset('basenji.n.01'), Synset('corgi.n.01'), Synset('cur.n.01'),
 Synset('dalmatian.n.02'), Synset('great_pyrenees.n.01'),
 Synset('griffon.n.02'), Synset('hunting_dog.n.01'), Synset('lapdog.n.01'),
 Synset('leonberg.n.01'), Synset('mexican_hairless.n.01'),
 Synset('newfoundland.n.01'), Synset('pooch.n.01'), Synset('poodle.n.01'), ...]
>>> list(dog.closure(hyper))
[Synset('canine.n.02'), Synset('domestic_animal.n.01'), Synset('carnivore.n.01'),
Synset('animal.n.01'), Synset('placental.n.01'), Synset('organism.n.01'),
Synset('mammal.n.01'), Synset('living_thing.n.01'), Synset('vertebrate.n.01'),
Synset('whole.n.02'), Synset('chordate.n.01'), Synset('object.n.01'),
Synset('physical_entity.n.01'), Synset('entity.n.01')]
```

Regression Tests

Bug 85: morphy returns the base form of a word, if it's input is given as a base form for a POS for which that word is not defined:

```
>>> wn.synsets('book', wn.NOUN)
[Synset('book.n.01'), Synset('book.n.02'), Synset('record.n.05'), Synset('script.n.01'), Synset('ledger.n.01'), Synset('book.n.06
>>> wn.synsets('book', wn.ADJ)
[]
>>> wn.morphy('book', wn.NOUN)
'book'
>>> wn.morphy('book', wn.ADJ)
```

Bug 160: wup_similarity breaks when the two synsets have no common hypernym

```
>>> t = wn.synsets('picasso')[0]
>>> m = wn.synsets('male')[1]
>>> t.wup_similarity(m)  # doctest: +ELLIPSIS
0.631...
```

```
>>> t = wn.synsets('titan')[1]
>>> s = wn.synsets('say', wn.VERB)[0]
>>> print(t.wup_similarity(s))
None
```

Bug 21: "instance of" not included in LCS (very similar to bug 160)

```
>>> a = wn.synsets("writings")[0]
>>> b = wn.synsets("scripture")[0]
>>> brown_ic = wordnet_ic.ic('ic-brown.dat')
>>> a.jcn_similarity(b, brown_ic)  # doctest: +ELLIPSIS
0.175...
```

Bug 221: Verb root IC is zero

```
>>> from nltk.corpus.reader.wordnet import information_content
>>> s = wn.synsets('say', wn.VERB)[0]
>>> information_content(s, brown_ic)  # doctest: +ELLIPSIS
4.623...
```

Bug 161: Comparison between WN keys/lemmas should not be case sensitive

```
>>> k = wn.synsets("jefferson")[0].lemmas()[0].key()
>>> wn.lemma_from_key(k)
Lemma('jefferson.n.01.Jefferson')
>>> wn.lemma_from_key(k.upper())
Lemma('jefferson.n.01.Jefferson')
```

Bug 99: WordNet root_hypernyms gives incorrect results

```
>>> from nltk.corpus import wordnet as wn
>>> for s in wn.all_synsets(wn.NOUN):
...     if s.root_hypernyms()[0] != wn.synset('entity.n.01'):
...         print(s, s.root_hypernyms())
...
>>>
```

Bug 382: JCN Division by zero error

```
>>> tow = wn.synset('tow.v.01')
>>> shlep = wn.synset('shlep.v.02')
>>> from nltk.corpus import wordnet_ic
>>> brown_ic =  wordnet_ic.ic('ic-brown.dat')
>>> tow.jcn_similarity(shlep, brown_ic)  # doctest: +ELLIPSIS
1...e+300
```

Bug 428: Depth is zero for instance nouns

```
>>> s = wn.synset("lincoln.n.01")
>>> s.max_depth() > 0
True
```

Bug 429: Information content smoothing used old reference to all_synsets

```
>>> genesis_ic = wn.ic(genesis, True, 1.0)
```

Bug 430: all_synsets used wrong pos lookup when synsets were cached

```
>>> for ii in wn.all_synsets(): pass
>>> for ii in wn.all_synsets(): pass
```

Bug 470: shortest_path_distance ignored instance hypernyms

```
>>> google = wordnet.synsets("google")[0]
>>> earth = wordnet.synsets("earth")[0]
>>> google.wup_similarity(earth)  # doctest: +ELLIPSIS
0.1...
```

Bug 484: similarity metrics returned -1 instead of None for no LCS

```
>>> t = wn.synsets('fly', wn.VERB)[0]
>>> s = wn.synsets('say', wn.VERB)[0]
>>> print(s.shortest_path_distance(t))
None
>>> print(s.path_similarity(t, simulate_root=False))
None
>>> print(s.lch_similarity(t, simulate_root=False))
None
>>> print(s.wup_similarity(t, simulate_root=False))
None
```

Bug 427: "pants" does not return all the senses it should

```
>>> from nltk.corpus import wordnet
>>> wordnet.synsets("pants",'n')
[Synset('bloomers.n.01'), Synset('pant.n.01'), Synset('trouser.n.01'), Synset('gasp.n.01')]
```

Bug 482: Some nouns not being lemmatised by WordNetLemmatizer().lemmatize

```
>>> from nltk.stem.wordnet import WordNetLemmatizer
>>> WordNetLemmatizer().lemmatize("eggs", pos="n")
'egg'
>>> WordNetLemmatizer().lemmatize("legs", pos="n")
'leg'
```

Bug 284: instance hypernyms not used in similarity calculations

```
>>> wn.synset('john.n.02').lch_similarity(wn.synset('dog.n.01'))  # doctest: +ELLIPSIS
1.335...
>>> wn.synset('john.n.02').wup_similarity(wn.synset('dog.n.01'))  # doctest: +ELLIPSIS
```

```
0.571...
>>> wn.synset('john.n.02').res_similarity(wn.synset('dog.n.01'), brown_ic)  # doctest: +ELLIPSIS
2.224...
>>> wn.synset('john.n.02').jcn_similarity(wn.synset('dog.n.01'), brown_ic)  # doctest: +ELLIPSIS
0.075...
>>> wn.synset('john.n.02').lin_similarity(wn.synset('dog.n.01'), brown_ic)  # doctest: +ELLIPSIS
0.252...
>>> wn.synset('john.n.02').hypernym_paths()  # doctest: +ELLIPSIS
[[Synset('entity.n.01'), ..., Synset('john.n.02')]]
```

Issue 541: add domains to wordnet

```
>>> wn.synset('code.n.03').topic_domains()
[Synset('computer_science.n.01')]
>>> wn.synset('pukka.a.01').region_domains()
[Synset('india.n.01')]
>>> wn.synset('freaky.a.01').usage_domains()
[Synset('slang.n.02')]
```

Issue 629: wordnet failures when python run with -O optimizations

```
>>> # Run the test suite with python -O to check this
>>> wn.synsets("brunch")
[Synset('brunch.n.01'), Synset('brunch.v.01')]
```

Issue 395: wordnet returns incorrect result for lowest_common_hypernyms of chef and policeman

```
>>> wn.synset('policeman.n.01').lowest_common_hypernyms(wn.synset('chef.n.01'))
[Synset('person.n.01')]
```