PyMedTermino

Jean-Baptiste Lamy

13 novembre 2013

1 Introduction

PyMedTermino (Terminologies Médicales en Python) est un module Python permettant d'accéder facilement aux principales terminologies médicales en Python. Les terminologies suivantes sont proposées :

- La SNOMED CT
- La CIM10
- L'UMLS
- Les icônes VCM (une terminologie à base d'icônes développée à l'université Paris 13)

Les principales fonctionnalités de PyMedTermino sont les suivantes :

- Une API unique permettant d'accéder à toutes les terminologies
- Recherche textuelle optimisée
- Accès aux libellés, synonymes et traductions
- Gestion des concepts et des relations entre concepts
- Correspondance entre terminologies, via l'UMLS ou via des fichiers de correspondances manuelles

Pour la SNOMED CT et la CIM10, les données ne sont pas incluses (car elles ne sont pas librement redistribuables) mais peuvent être téléchargées au format XML gratuitement. PyMedTermino contient des scripts pour exporter ces données dans des bases de données SQLite3.

Pour l'UMLS, les données ne sont pas incluses (pour les mêmes raisons, et car elles sont très volumineuses). PyMedTermino peut en revanche se connecter à un serveur MySQL contenant les bases UMLS telles que fournies par la NLM.

Pour les icônes VCM, l'ensemble des terminologies est fourni sous forme d'ontologie OWL et de bases de données SQLite3. En revanche, les images des icônes ne sont pas incluses et doivent être téléchargées séparément (ou bien vous pouvez utiliser le serveur iconique pour générer les icônes) : PyMedTermino ne comprend que la partie "terminologique" de VCM.

PyMedTermino a été réalisé au laboratoire LIM&BIO, Université Paris 13, Sorbonne Paris Cité, par Jean-Baptiste Lamy. PyMedTermino est disponible sous licence GNU LGPL.

Voici un exemple des possibilités de PyMedTermino :

```
>>> SNOMEDCT.search("tachycardia*")
[SNOMEDCT[3424008]  # Tachycardia (finding)
, SNOMEDCT[4006006]  # Fetal tachycardia affecting management of mother (disorder)
, SNOMEDCT[6456007]  # Supraventricular tachycardia (disorder)
...]

>>> SNOMEDCT[3424008].parents
[SNOMEDCT[301113001]  # Finding of heart rate (finding)
]

>>> SNOMEDCT[3424008].children
[SNOMEDCT[11092001]  # Sinus tachycardia (finding)
, SNOMEDCT[278086000]  # Baseline tachycardia (finding)
, SNOMEDCT[162992001]  # On examination - pulse rate tachycardia (finding)
...]

>>> list(SNOMEDCT[3424008].ancestors_no_double())
[SNOMEDCT[301113001]  # Finding of heart rate (finding)
, SNOMEDCT[106066004]  # Cardiac rhythm AND/OR rate finding (finding)
, SNOMEDCT[250171008]  # Clinical history and observation findings (finding)
, SNOMEDCT[404684003]  # Clinical finding (finding)
, SNOMEDCT[138875005]  # SNOMED CT Concept (SNOMED RT+CTV3)
```

```
...]

>>> SNOMEDCT[3424008].relations
set(['INVERSE_has_definitional_manifestation', 'finding_site', 'inter-
prets', 'has_interpretation', 'INVERSE_associated_with'])

>>> SNOMEDCT[3424008].finding_site
[SNOMEDCT[24964005]  # Cardiac conducting system structure (body structure)
]

>>> SNOMEDCT[3424008] >> VCM   # Maps the SNOMED CT concept to VCM icon
Concepts([
  VCM[u"current--hyper--heart_rhythm"]  #
])
```

2 Installation

1. Extraire l'archive PyMedTermino-X.Y.tar.bz2
2. Après inscription auprès de la NLM, télécharger les données de la SNOMED CT :
   - http ://www.nlm.nih.gov/research/umls/licensedcontent/snomedctfiles.html
     (télécharger "SnomedCT_Release_INT_<date>.zip" et le décompresser)
   - http ://www.nlm.nih.gov/research/umls/Snomed/core_subset.html
     (télécharger "SNOMEDCT_CORE_SUBSET_<date>" et le décompresser)
3. Après inscription, télécharger les données de la CIM10 ainsi que les traductions :
   - http ://apps.who.int/classifications/apps/icd/ClassificationDownload/DLArea/Download.aspx
     (télécharger la CIM10 au format "ClaML" et la dézipper)
   - http ://www.icd10.ch/
     (optionnel, pour les traductions françaises et allemandes ; télécharger la CIM10 au format XML et la dézipper)
4. Éditer le fichier setup.py et renseigner les 4 chemins où vous avez décompressés les données, par exemple :

   ```
   SNOMEDCT_DIR = "/home/jiba/telechargements/base_med/SnomedCT_Release_INT_20130731"
   SNOMEDCT_CORE_FILE = "/home/jiba/telechargements/base_med/SNOMEDCT_CORE_SUBSET_201308.txt"
   ICD10_DIR = "/home/jiba/telechargements/base_med/icd10"
   CIM10_DIR = "/home/jiba/telechargements/base_med/cim10"
   ```

   Note : vous pouvez laisser vide les chemins si vous ne souhaitez pas installer les terminologies correspondantes.
5. Compiler PyMedTermino et convertir les données télécharger en bases de données SQL SQLite3 :

   ```
   python setup.py build
   ```

   Attention : la création des bases de données nécessite un espace disque important (˜1-2 Go).
6. Obtenir les droits administrateurs. Sous Linux, selon votre distribution, utilisez l'une d'une commande suivante :

   ```
   su # Mageia,...
   sudo -i # Ubuntu, Linux Mint,...
   ```

7. Installer PyMedTermino :

   ```
   python setup.py install
   ```

8. Nettoyer le répertoire d'installation (optionnel, mais cela permet de libérer un espace disque important !) :

   ```
   python setup.py clean
   ```

2.1 Problèmes fréquents

2.1.1 OperationalError : no such module : fts4

Sous Windows, si vous rencontrez ce problème à l'installation, il vous faut mettre à jour la DLL Sqlite3. Pour cela, télécharger la dernière version à partir de http ://www.sqlite.org/download.html, et remplacer la DLL dans le répertoire Python27/DLLs par la version téléchargée.

2.1.2 IOError : [Errno 22] Invalid argument

Sous Windows, on obtient cette erreur si les fichiers volumineux de la SNOMED CT sont sur un répertoire partagé. C'est une limitation lié au système de Microsoft, il faut donc impérativement avoir les fichier de la SNOMED en local.

3 Chargement des modules et paramètres globaux

```
>>> import pymedtermino
>>> pymedtermino.LANGUAGE = "fr"
>>> pymedtermino.REMOVE_SUPPRESSED_CONCEPTS = 1
```

Les paramètres globaux suivants sont disponibles :

- pymedtermino.DATA_DIR : indique le répertoire où sont présents les fichiers de base de données SQLite3 contenant les terminologies. Valeur par défaut : le répertoire de PyMedTermino.
- pymedtermino.LANGUAGE : indique la langue utiliser par défaut pour afficher les termes, lorsque plusieurs langues sont disponibles. Si la langue demandé n'est pas disponible, l'anglais est utilisé à défaut. ATTENTION : ce paramètre doit être renseigné AVANT de charger les terminologies. Valeur par défaut : "en" (anglais).
- pymedtermino.REMOVE_SUPPRESSED_CONCEPTS : indique s'il faut retirer ou non les concepts marqués comme supprimés ou dépréciés dans les terminologies. Valeur par défaut : 1 (vrai).
- pymedtermino.REMOVE_SUPPRESSED_TERMS : indique s'il faut retirer ou non les termes (=libellés textuels) marqués comme supprimés ou dépréciés dans les terminologies. Valeur par défaut : 1 (vrai).
- pymedtermino.REMOVE_SUPPRESSED_RELATIONS : indique s'il faut retirer ou non les relations marquées comme supprimées ou dépréciées dans les terminologies. Valeur par défaut : 1 (vrai).

4 SNOMED CT

4.1 Chargement des modules

Ensuite, il est possible d'importer en Python la SNOMED CT :

```
>>> from pymedtermino import *
>>> from pymedtermino.snomedct import *
```

4.2 Concepts

L'objet SNOMEDCT représente la terminologie SNOMED CT. Un concept SNOMED CT peut être obtenu à partir de son code (ici nous allons prendre comme exemple le code 302509004, correspondant au cœur) en indexant cet objet :

```
>>> concept = SNOMEDCT[302509004]
>>> concept
SNOMEDCT[302509004]  # Entire heart (body structure)
```

La méthode has_concept() permet de vérifier si un code correspond à un concept ou non :

```
>>> SNOMEDCT.has_concept("invalid_code")
False
```

Chaque concept possède un code et un terme (= libellé correspondant au terme préféré) :

```
>>> concept.code
302509004

>>> concept.term
u'Entire heart (body structure)'
```

La SNOMED CT propose aussi des termes synonymes (notez le "s" sur "terms") :

```
>>> concept.terms
[u'Heart', u'Entire heart', u'Entire heart (body structure)']
```

4.3 Recherche textuelle

La méthode search() permet d'effectuer une recherche textuelle, parmi les libellées des concepts et leurs synonymes :

```
>>> SNOMEDCT.search("Cardiac structure")
[ SNOMEDCT[80891009] # Heart structure (body structure)
, SNOMEDCT[308793001] # Embryonic cardiac structure (body structure)
...]
```

La recherche textuelle utilise le moteur FTS de SQLite, il est donc possible d'utiliser les fonctionnalités offertes par ce moteur. Par exemple pour rechercher tous les mots commençant par un préfixe donné :

```
>>> SNOMEDCT.search("osteo*")
[ SNOMEDCT[1551001]  # Osteomyelitis of femur (disorder)
, SNOMEDCT[4598005]  # Osteomalacia (disorder)
...]
```

4.4 Relations est-un : concepts parents et enfants

Les attributs “parents” et “children” permettent d’obtenir la liste des concepts parents et des concepts enfants (c’est-à-dire ceux reliés au concept par des relations est-un) :

```
>>> concept.parents
[SNOMEDCT[116004006]  # Hollow viscus (body structure)
, SNOMEDCT[80891009]  # Heart structure (body structure)
, SNOMEDCT[187639008]  # Entire thoracic viscus (body structure)
]
>>> concept.children
[SNOMEDCT[195591003]  # Entire transplanted heart (body structure)
]
```

Les méthodes ancestors() et descendants() permettent de parcourir les concepts ancêtres (les parents, les parents des parents, etc) et les concepts descendants (les enfants, les enfants des enfants, etc) :

```
>>> for ancestor in concept.ancestors(): print ancestor
SNOMEDCT[116004006]  # Hollow viscus (body structure)
SNOMEDCT[118760003]  # Entire viscus (body structure)
SNOMEDCT[272625005]  # Entire body organ (body structure)
[...]
```

Les méthodes ancestors() et descendants() retournent des générateurs Python, pour obtenir la liste des ancêtres ou des descendants il faut utiliser la fonction list() :

```
>>> concept.ancestors()
<generator object ancestors at 0xb3f734c>

>>> list(concept.ancestors())
[SNOMEDCT[116004006]  # Hollow viscus (body structure)
, SNOMEDCT[118760003]  # Entire viscus (body structure)
, SNOMEDCT[272625005]  # Entire body organ (body structure)
,...]

>>> list(concept.descendants())
[SNOMEDCT[195591003]  # Entire transplanted heart (body structure)
]
```

Les méthodes ancestors_no_double() et descendants_no_double() fonctionnent de la même manière mais en éliminant les doublons. Les méthodes self_and_ancestors() et self_and_descendants() fonctionnent de la même manière mais retournent aussi le concept de départ lui-même. Les méthodes self_and_ancestors_no_double() et self_and_descendants_no_double() combinent les deux comportements.

Enfin, la méthode is_a() permet de tester si un concept est un descendant d’un autre concept :

```
>>> concept.is_a(SNOMEDCT[272625005])
True
```

4.5 Relations partie-de

Les attributs “part_of” et “INVERSE_part_of” permettent d’accéder aux concepts partie ou tout :

```
>>> concept.part_of
[SNOMEDCT[362010009] # Entire heart AND pericardium (body structure)
]

>>> concept.INVERSE_part_of
[SNOMEDCT[102298001] # Structure of chordae tendineae cordis (body structure)
, SNOMEDCT[181285005] # Entire heart valve (body structure)
, SNOMEDCT[181288007] # Entire tricuspid valve (body structure)
, SNOMEDCT[181293005] # Entire cardiac wall (body structure)
,...]
```

Les méthodes ancestor_parts() et descendant_parts() retournent un générateur Python permettant de parcourir les super- ou sous-parties du concept :

```
>>> list(concept.ancestor_parts())
[SNOMEDCT[362010009] # Entire heart AND pericardium (body structure)
, SNOMEDCT[362688008] # Entire middle mediastinum (body structure)
, SNOMEDCT[181217005] # Entire mediastinum (body structure)
```

```
, SNOMEDCT[302551006] # Entire thorax (body structure)
,...]

>>> list(concept.descendant_parts())
[SNOMEDCT[181285005]  # Entire heart valve (body structure)
, SNOMEDCT[192664000]  # Entire cardiac valve leaflet (body structure)
, SNOMEDCT[192747009]  # Structure of cardiac valve cusp (body structure)
,...]
```

Enfin, la méthode is_part_of() permet de tester si un concept est une partie d'un autre concept (de manière récursive) :

```
>>> concept.is_part_of(SNOMEDCT[91744000])
False
```

4.6 Autres relations

L'attribut "relations" permet d'obtenir la liste des types de relations disponibles pour ce concept. Les relations est-un (is_a) ne sont jamais incluses dans "relations", elles sont gérées via les attributs "parents" et "children" vus précédemment, en revanche les relations partie-de y figurent. Les relations inverses sont préfixées par "INVERSE_".

```
>>> concept = SNOMEDCT[3424008]
>>> concept
SNOMEDCT[3424008] # Tachycardia (finding)

>>> concept.relations
set([u'INVERSE_has_definitional_manifestation', u'finding_site', u'inter-
prets', u'has_interpretation', u'INVERSE_associated_with'])
```

Chaque relation correspond à un attribut du concept, qui retourne une liste avec le ou les valeurs correspondantes :

```
>>> concept.finding_site
[SNOMEDCT[24964005] # Cardiac conducting system structure (body structure)
]
>>> concept.interprets
[SNOMEDCT[364075005]  # Heart rate (observable entity)
]
>>> concept.INVERSE_has_definitional_manifestation
[ SNOMEDCT[413342000]  # Neonatal tachycardia (disorder)
, SNOMEDCT[195069001]  # Paroxysmal atrial tachycardia (disorder)
, SNOMEDCT[195070000]  # Paroxysmal atrioventricular tachycardia (disorder)
,...]
```

4.7 Groupes de relations

Dans la SNOMED CT, les relations peuvent être regroupées en groupes. L'attribut "groups" permet d'obtenir la liste des groupes de relation. Il est ensuite possible d'accéder aux relations du groupe comme pour un concept.

```
>>> SNOMEDCT[186675001]
SNOMEDCT[186675001]  # Viral pharyngoconjunctivitis (disorder)

>>> SNOMEDCT[186675001].groups
[<Group associated_morphology Inflammation (morphologic abnormality); finding_site Conjuncti-
val structure (body structure)>, <Group associated_morphology Inflammation (morphologic ab-
normality); finding_site Pharyngeal structure (body structure)>]

>>> SNOMEDCT[186675001].groups[0].relations
set([u'associated_morphology', u'finding_site'])
>>> SNOMEDCT[186675001].groups[0].finding_site
Concepts([
  SNOMEDCT[29445007]  # Conjunctival structure (body structure)
])
>>> SNOMEDCT[186675001].groups[0].associated_morphology
Concepts([
  SNOMEDCT[23583003]  # Inflammation (morphologic abnormality)
])
```

Les relations qui n'appartiennent à aucun groupe sont réunies dans un groupe "hors-groupe" (qui ne figure pas dans la liste "groups").

```
>>> SNOMEDCT[186675001].out_of_group
<Group causative_agent Virus (organism); pathological_process Infectious process (quali-
fier value)>
```

4.8 Parcourir la SNOMED CT

Pour obtenir les premiers niveaux de la terminologie (= les concepts racines), il faut utiliser la méthode first_levels() :

```
>>> SNOMEDCT.first_levels()
[ SNOMEDCT[123037004] # Body structure (body structure)
, SNOMEDCT[404684003] # Clinical finding (finding)
, SNOMEDCT[308916002] # Environment or geographical location (environment / location)
,...]
```

La méthode all_concepts() retourne un générateur Python qui parcourt tous les concepts de la SNOMED CT.

```
>>> for concept in SNOMEDCT.all_concepts(): [...]
```

La méthode all_concepts_no_double() fonctionne de la même manière mais élimine les doublons.

```
>>> for concept in SNOMEDCT.all_concepts_no_double(): [...]
```

4.9 CORE Problem List

La CORE Problem List est un sous-ensemble de la SNOMED CT approprié pour le codage de l'information clinique. L'attribut "is_in_core" permet de savoir si un concept appartient à la CORE Problem List :

```
>>> concept.is_in_core
1
```

Il est aussi possible de parcourir tous les concepts de la CORE Problem List :

```
>>> for core_concept in SNOMEDCT.CORE_problem_list(): [...]
```

4.10 Signes cliniques associées à un concept

La méthode associated_clinical_findings() permet de lister tous les signes cliniques associés à un concept de structure anatomique (*body structure*) ou de morphologie, y compris leurs descendants et leur parties descendantes. Par exemple pour lister toutes les maladies des structures cardiaques :

```
>>> SNOMEDCT[80891009]
SNOMEDCT[80891009]  # Heart structure (body structure)

>>> SNOMEDCT[80891009].associated_clinical_findings()
Concepts([
  SNOMEDCT[250981008]  # Abnormal aortic cusp (disorder)
, SNOMEDCT[250982001]  # Commissural fusion of aortic cusp (disorder)
, SNOMEDCT[250984000]  # Torn aortic cusp (disorder)
,...]
```

5 CIM10

5.1 Chargement des modules

```
>>> from pymedtermino import *
>>> from pymedtermino.icd10 import *
```

5.2 Concepts

L'objet ICD10 permet d'accéder aux concepts de la CIM10. Cet objet fonctionne de manière très proche de la terminologie SNOMED CT décrite précédemment.

```
>>> ICD10["E10"]
ICD10[u"E10"]  # diabète sucré insulino-dépendant

>>> ICD10["E10"].parents
[ICD10[u"E10-E14"]  # diabète sucré
]
```

```
>>> list(ICD10["E10"].ancestors())
[ ICD10[u"E10-E14"]  # diabète sucré
, ICD10[u"IV"]  # maladies endocriniennes, nutritionnelles et métaboliques
]
```

La CIM10 étant monoaxiale, la liste parents contient au plus un seul concept parent.

5.3 Traduction

La CIM10 est disponible en plusieurs langues. La méthode get_translation() permet d'obtenir la traduction dans une langue donnée :

```
>>> print(ICD10["E10"].get_translation("fr"))
diabète sucré insulino-dépendant

>>> print(ICD10["E10"].get_translation("en"))
Insulin-dependent diabetes mellitus
```

La langue utilisée par défaut est défini par pymedtermino.LANGUAGE (qui doit être défini AVANT de charger les concepts).

5.4 Relations

Les relations incluent les relations d'inclusion et d'exclusion de la CIM10.

```
>>> ICD10["E10"].relations
set([u'inclusion', u'exclusion', u'modifierlink'])

>>> ICD10["E10"].exclusion
[Text(ICD10[u"E10"]  # diabète sucré insulino-dépendant
, 'exclusion', u'diabetes mellitus (in) malnutrition-related E12.-
', 0, ICD10[u"E12"]  # diabète sucré de malnutrition
)...]
```

6 UMLS

6.1 Chargement des modules

```
>>> from pymedtermino import *
>>> from pymedtermino.umls import *
```

Une fois les modules importées, il faut se connecter à la base de données MySQL contenant les données de l'UMLS, de la manière suivante :

```
>>> connect_to_umls_db(hôte, utilisateur, mot_de_passe, nom_de_la_base = "umls", en-
codage = "latin1")
```

Hôte, utilisateur, mot_de_passe doivent être précisés.

6.2 Concepts UMLS (CUI)

Dans UMLS, les CUI correspondent à des concepts : un même concept rassemble des termes et des codes équivalent de différentes terminologies.

PyMedTermino permet d'accéder aux CUI via la terminologie UMLS_CUI :

```
>>> UMLS_CUI[u"C0085580"]
UMLS_CUI[u"C0085580"] # Hypertension artérielle essentielle (MDR-
JPN, SNOMEDCT, ICD10, BI, CCS, MDRPOR, COSTAR, ICD10DUT, KCD5, RCD, MDRGER, AOD, MDRFRE, MDR-
CZE, SCTSPA, DMDICD10, ICPC2P, OMIM, MDRITA, MDR, MEDCIN, ICD10CM, MDR-
DUT, ICD10AM, MTH, CSP, MDRSPA, SNM, DXP, NCI, PSY, SNMI, ICD9CM, CCPSS)

>>> UMLS_CUI[u"C0085580"].term
u'Hypertension art\xe9rielle essentielle'

>>> UMLS_CUI[u"C0085580"].terms
['Hypertension art\xe9rielle essentielle', 'Hypertension primitive', 'Hypertension essen-
tielle, non pr\xe9cis\xe9e', 'Hypertension essentielle non pr\xe9cis\xe9e']

>>> UMLS_CUI[u"C0085580"].original_terminologies
```

```
set(['MDRJPN', 'SNOMEDCT', 'ICD10', 'BI', 'CCS', 'MDRPOR', 'COSTAR', 'ICD10DUT', 'KCD5', 'R-
CD', 'MDRGER', 'AOD', 'MDRFRE', 'MDRCZE', 'SCTSPA', 'DMDICD10', 'ICPC2P', 'OMIM', 'MDRI-
TA', 'MDR', 'MEDCIN', 'ICD10CM', 'MDRDUT', 'ICD10AM', 'MTH', 'CSP', 'MDRSPA', 'S-
NM', 'DXP', 'NCI', 'PSY', 'SNMI', 'ICD9CM', 'CCPSS'])
```

Il est possible de manipuler les relations des CUI de la même manière que pour les concepts SNOMED CT (voir section 4.6), par exemple :

```
>>> UMLS_CUI[u"C0085580"].relations
set(['has_finding_site', 'INVERSE_translation_of', 'SIB', 'IN-
VERSE_has_alias', 'may_be_a', None, 'RQ', 'INVERSE_mapped_from',...])

>>> UMLS_CUI[u"C0085580"].has_finding_site
[UMLS_CUI[u"C0459964"]  # Systemic arterial structure (RCD, SCTSPA, SNOMEDCT)
```

6.3 Concept UMLS issus des terminologies sources (AUI)

La terminologie UMLS_AUI permet d'accéder aux atomes de l'UMLS. Un atome UMLS correspond à un concept dans une terminologie source donnée ; "diabète de type 2 dans la CIM10" est un atome différent de "diabète de type 2 dans la SNOMED CT".

```
>>> UMLS_AUI[u"A0930328"]
UMLS_AUI[u"A0930328"] # Essential (primary) hypertension (ICD10)

>>> UMLS_AUI[u"A0930328"].original_terminologies
set(['ICD10'])
```

6.4 Extraction de terminologie de l'UMLS

PyMedTermino permet d'extraire des terminologies de l'UMLS, et de les utiliser avec les codes des terminologies sources (plutôt que les AUI), par exemple pour extraire la SNOMED CT, la CIM10 et la CISP 2 :

```
>>> UMLS_SNOMEDCT  = UMLS_AUI.extract_terminology("SNOMEDCT", has_int_code = 1)
>>> UMLS_ICD10     = UMLS_AUI.extract_terminology("ICD10")
>>> UMLS_ICPC2EENG = UMLS_AUI.extract_terminology("ICPC2EENG")
```

Le premier paramètre de la fonction UMLS_AUI.extract_terminology() est le nom de la terminologie à extraire (que l'on peut trouver dans la liste des sources de l'UMLS). Le paramètre optionnel "has_int_code = 1" permet d'indiquer que les codes de la terminologie source sont numériques, ce qui évite ensuite d'avoir à les mettre entre guillemets.

Les terminologies extraites peuvent ensuite être utilisées :

```
>>> UMLS_ICD10["I10"]
UMLS_ICD10[u"I10"]  # Essential (primary) hypertension (ICD10)
```

Il est possible d'accéder aux relations (lorsqu'elles existent) de la même manière que précédemment.

6.5 Correspondance entre terminologies de l'UMLS

PyMedTermino défini automatiquement des correspondances entre les terminologies extraites de l'UMLS, par exemple :

```
>>> UMLS_ICD10["I10"] >> UMLS_SNOMEDCT
Concepts([
  UMLS_SNOMEDCT[u"59621000"]  # Essential hypertension (SNOMEDCT)
])
```

Pour plus d'information sur les correspondances, voir la section 8.

7 VCM

7.1 Chargement des modules

```
>>> from pymedtermino import *
>>> from pymedtermino.vcm import *
```

Les bases de données décrivant les terminologies VCM sont incluses dans PyMedTermino.

7.2 Icônes VCM

L'objet VCM permet d'accéder aux icônes VCM, identifiées par leur code, en français ou en anglais :

```
>>> icon = VCM["en_cours--patho--coeur"]
>>> icon = VCM["current--patho--heart"]
>>> icon =  VCM["en_cours--patho-vaisseau--coeur--traitement--medicament--rien--rien"]
```

Le code d'icône inclut jusqu'à 7 composantes, séparé par deux tirets (–) :

1. La couleur centrale
2. Le ou les modificateurs de forme (séparés par un seul tiret si plusieurs)
3. Le pictogramme central
4. La couleur en exposant
5. Le pictogramme en exposant
6. Le pictogramme en second exposant
7. L'ombre

Les valeurs possibles pour chaque composante sont listées dans le lexique graphique (voir le lexique des pictogrammes VCM, ou la terminologie VCM_LEXICON ci-dessous). Les composantes absentes dans le code de l'icône sont remplacées par la valeur rien / empty.

Des attributs permettent de récupérer les différentes composantes d'une icône :

```
>>> icon.central_color
VCM_LEXICON[496] # Red_color
>>> icon.modifiers
Concepts([
  VCM_LEXICON[536]  # Modifier_vessel
, VCM_LEXICON[504]  # Modifier_patho
])
>>> icon.central_pictogram
VCM_LEXICON[549]  # Pictogramme_heart
>>> icon.central_pictogram.text_code
heart

>>> icon.top_right_color
VCM_LEXICON[690]  # Green_color
>>> icon.top_right_pictogram
VCM_LEXICON[697]  # Drug_top_right_pictogram
>>> icon.second_top_right_pictogram
VCM_LEXICON[718]  # No_second_top_right_pictogram
>>> icon.shadow
VCM_LEXICON[722]  # No_shadow
```

L'attribut "lexs" permet d'obtenir l'ensemble des composantes :

```
>>> icon.lexs
Concepts([
  VCM_LEXICON[536]  # Modifier_vessel
, VCM_LEXICON[549]  # Pictogramme_heart
, VCM_LEXICON[722]  # No_shadow
, VCM_LEXICON[496]  # Red_color
, VCM_LEXICON[504]  # Modifier_patho
, VCM_LEXICON[718]  # No_second_top_right_pictogram
, VCM_LEXICON[697]  # Drug_top_right_pictogram
, VCM_LEXICON[690]  # Green_color
])
```

Les attributs suivants permettent d'obtenir les modificateurs d'une catégorie précise : modificateur pathologique ou physiologique, étiologique,... :

```
>>> icon.physio
>>> icon.patho
>>> icon.etiology
>>> icon.quantitative
>>> icon.process
>>> icon.transverse
```

L'attribut "consistent" permet de savoir si l'icône est consistante ou non (vis-à-vis de l'ontologie des icônes VCM, décrite dans l'article : J-B Lamy et al., Validating the semantics of a medical iconic language using ontological reasoning[1]) :

```
>>> icon.consistent
True
```

7.3 Lexique graphique

La terminologie VCM_LEXICON décrit le lexique graphique des primitives des icônes VCM : pictogrammes, couleurs et formes. Chaque primitive est identifié par un code numérique arbitraire, par exemple pour le pictogramme du cœur :

```
>>> heart = VCM_LEXICON[549]
>>> heart
VCM_LEXICON[549] # Pictogramme_heart
```

Chaque concept du lexique possède aussi des codes textuels (plus facile à retenir que le code numérique, disponible en français et en anglais), et une catégorie :

```
>>> heart.text_code
u'coeur'
>>> heart.text_codes
[u'heart', u'coeur']
>>> heart.category
2
```

Les catégories correspondent aux différentes parties des icônes VCM :

0 Couleur centrale

1 Modificateur de forme

2 Pictogramme central

3 Couleur en exposant

4 Pictogramme en exposant

5 Pictogramme en second exposant

6 Ombre

Il est aussi possible d'obtenir un concept du lexique à partir de sa catégorie et de son code textuel :

```
>>> VCM_LEXICON[2, "heart"]
VCM_LEXICON[549] # Pictogramme_heart
```

Les relations sont gérés comme d'ordinaire dans PyMedTermino (voir section sur la SNOMED CT : parents, children, is_a(), ancestors(), descendants(),...). De plus la relation graphical_is_a indique les autres éléments du lexique qui sont réutilisés. Par exemple le pictogramme du rythme cardiaque reprend le pictogramme du cœur :

```
>>> heart_rhythm = VCM_LEXICON[2, "heart_rhythm"]
>>> heart_rhythm.graphical_is_a
[VCM_LEXICON[549]  # Pictogramme_heart
]
```

Les attributs "graphical_children" et "graphical_parents" permettent d'obtenir la liste des éléments du lexique qui réutilisent ou qui sont réutilisés par un autre.

7.3.1 Créer une icône VCM à partir d'éléments du lexique

Un ensemble de concepts du lexique peut être converti en icône VCM :

```
>>> Concepts([VCM_LEXICON[549], VCM_LEXICON[496], VCM_LEXICON[504]]) >> VCM
Concepts([
  VCM[u"en_cours--patho--coeur"]  #
])
```

1. J-B Lamy et al., Validating the semantics of a medical iconic language using ontological reasoning, Journal of Biomedical Informatics 2013, 46(1) :56-67
http ://www.sciencedirect.com/science/article/pii/S153204641200130X

7.4 Concepts médicaux

VCM_CONCEPT est une terminologie qui représente les concepts médicaux de VCM. Chaque concept médical est défini par un code arbitraire, par exemple pour le cœur :

```
>>> heart = VCM_CONCEPT[266]
>>> heart
VCM_CONCEPT[266] # Cardiac_structure
```

Les relations sont gérés comme d'ordinaire dans PyMedTermino (voir section sur la SNOMED CT : parents, children, is_a(), ancestors(), descendants(), relations...).

VCM_CONCEPT_MONOAXIAL est une terminologie identique à VCM_CONCEPT, mais monoaxiale. Les concepts sont donc les mêmes, mais avec au maximum un seul parent par concept. Cette terminologie est principalement utilisé en interne pour relier VCM_CONCEPT (multiaxial) à VCM_LEXICON (monoaxial).

8 Correspondances

Une correspondance (mapping) permet de transcoder un ou plusieurs concepts d'une terminologie source vers une terminologie destination. PyMedTermino utilise l'opérateur >> pour les correspondances, de la manière suivante :

```
concept(s) >> TERMINOLOGIE_DESTINATION
```

ou concept(s) peut être soit un concept de la terminologie source, soit un ensemble de concepts (voir section 9). L'opérateur >> retourne un ensemble de concepts dans la terminologie destination. Les opérateurs >> peuvent donc être chaînés :

```
concept(s) >> TERMINOLOGIE_INTERMEDIAIRE >> TERMINOLOGIE_DESTINATION
```

PyMedTermino inclut plusieurs correspondances, décrite dans les sous-sections suivantes.

8.1 Correspondances UMLS

8.1.1 UMLS_CUI <=> UMLS_AUI

PyMedTermino peut convertir les CUI en AUI et vice versa :

```
>>> UMLS_CUI[u"C0085580"] >> UMLS_AUI
Concepts([
  UMLS_AUI[u"A16015049"]  # Hypertension primitive (MDRFRE)
, UMLS_AUI[u"A11101884"]  # Hypertension essentielle, non précisée (MDRFRE)
, UMLS_AUI[u"A11089284"]  # Hypertension essentielle non précisée (MDRFRE)
...])
```

8.1.2 Terminologie extraite de l'UMLS <=> CUI ou AUI

PyMedTermino peut convertir les concepts des terminologies extraites de l'UMLS en CUI ou en AUI, et vice versa :

```
>>> UMLS_ICD10["I10"] >> UMLS_CUI
Concepts([
  UMLS_CUI[u"C0085580"]  # Hypertension artérielle essentielle (MDR-
JPN, SNOMEDCT, ICD10, BI, CCS, MDRPOR, COSTAR, ICD10DUT, KCD5, RCD, MDRGER, AOD, MDRFRE, MDR-
CZE, SCTSPA, DMDICD10, ICPC2P, OMIM, MDRITA, MDR, MEDCIN, ICD10CM, MDR-
DUT, ICD10AM, MTH, CSP, MDRSPA, SNM, DXP, NCI, PSY, SNMI, ICD9CM, CCPSS)
])
```

8.1.3 Terminologie extraite de l'UMLS <=> terminologie source

PyMedTermino peut convertir les concepts des terminologies extraites de l'UMLS vers la terminologie source, et vice versa :

```
>>> ICD10["I10"] >> UMLS_ICD10
Concepts([
  UMLS_ICD10[u"I10"]  # Essential (primary) hypertension (ICD10)
])
```

8.1.4 Terminologie extraite de l'UMLS <=> autre terminologie extraite de l'UMLS

PyMedTermino crée automatiquement des correspondances entre les terminologies extraites de l'UMLS avec UMLS_AUI.extract_terminology() :

```
>>> UMLS_ICD10["I10"] >> UMLS_SNOMEDCT
Concepts([
  UMLS_SNOMEDCT[u"59621000"]  # Essential hypertension (SNOMEDCT)
])
```

8.2 SNOMEDCT <=> VCM

Cette correspondance associe des icônes VCM aux concepts SNOMED CT. Elle a été construite de manière automatique à partir des correspondances SNOMEDCT <=> VCM_CONCEPT et VCM_CONCEPT <=> VCM_LEXICON (comme décrit dans l'article : J-B Lamy et al., A Semi-automatic Semantic Method for Mapping SNOMED CT Concepts to VCM Icons [2]).

```
>>> from pymedtermino.snomedct_2_vcm import *

>>> SNOMEDCT[3424008]
SNOMEDCT[3424008]  # Tachycardia (finding)

>>> SNOMEDCT[3424008] >> VCM
Concepts([
  VCM[u"en_cours--hyper--coeur_rythme"]  #
])
```

8.3 VCM_LEXICON => VCM

Un ensemble d'éléments du lexique peut être converti en icône VCM :

```
>>> Concepts([VCM_LEXICON[549], VCM_LEXICON[496], VCM_LEXICON[504]]) >> VCM
Concepts([
  VCM[u"en_cours--patho--coeur"]  #
])
```

8.4 VCM_CONCEPT <=> VCM_LEXICON

Cette correspondance permet de transformer un concept médical en élément du lexique VCM, et vice versa. Elle a été construite manuellement, et fait partie de l'ontologie des icônes VCM.

```
>>> VCM_CONCEPT[266] >> VCM_LEXICON
Concepts([
  VCM_LEXICON[549]  # Pictogramme_heart
])
>>> VCM_LEXICON[549] >> VCM_CONCEPT
Concepts([
  VCM_CONCEPT[266]  # Structure_cardiaque
, VCM_CONCEPT[102]  # Fonction_cardiaque
])
```

8.5 SNOMEDCT <=> VCM_CONCEPT

Cette correspondance associe les concepts SNOMED CT (principalement ceux de structures anatomiques et morphologies) aux concepts VCM. Elle a été construite manuellement.

```
>>> SNOMEDCT[302509004]
SNOMEDCT[302509004] # Entire heart (body structure)

>>> SNOMEDCT[302509004] >> VCM_CONCEPT
Concepts([
  VCM_CONCEPT[266] # Structure_cardiaque
, VCM_CONCEPT[239] # Région_du_thorax
])
```

2. J-B Lamy et al., A Semi-automatic Semantic Method for Mapping SNOMED CT Concepts to VCM Icons, Studies in health technology and informatics 2013, 192 :42-6
http ://ebooks.iospress.nl/publication/33954

8.6 Exemples

En chaînant plusieurs correspondances, il est possible de convertir un concept CIM10 en SNOMED CT via l'UMLS :

```
>>> ICD10["I10"] >> UMLS_ICD10 >> UMLS_SNOMEDCT >> SNOMEDCT
Concepts([
  SNOMEDCT[59621000]  # Essential hypertension (disorder)
])
```

Si vous souhaitez utiliser cette méthode par défaut pour les correspondances de la CIM10 vers la SNOMED CT, vous pouvez enregistrer cette correspondance de la manière suivante :

```
>>> (ICD10 >> UMLS_ICD10 >> UMLS_SNOMEDCT >> SNOMEDCT).register()

>>> ICD10["I10"] >> SNOMEDCT
Concepts([
  SNOMEDCT[59621000]  # Essential hypertension (disorder)
])
```

9 Ensembles de concepts

La classe Concepts() peut être utilisée pour représenter des ensembles de concepts (à la manière des set() de Python). Un tel ensemble ne peut contenir chaque concept qu'en un seul exemplaire, et il hérite des set() Python les méthodes permettant d'obtenir l'intersection, l'union, la différence, ..., de deux ensembles. Il propose aussi les méthodes suivantes :

- Concepts.find(concept_parent) : retourne le premier concept de l'ensemble qui est un descendant de concept_parent (y compris le concept_parent lui-même).
- Concepts.extract(concept_parent) : retourne tous les concepts de l'ensemble qui sont des descendants de concept_parent (y compris le concept_parent lui-même).
- Concepts.subtract(concept_parent) : retourne un nouvel ensemble après avoir retiré tous les concepts de l'ensemble qui sont des descendants de concept_parent (y compris le concept_parent lui-même).
- Concepts.subtract_update(concept_parent) : comme subtract(), mais modifie l'ensemble "en place".
- Concepts.imply(concepts2) : retourne vrai si chaque concept de l'ensemble est le descendant d'au moins un concept de l'ensemble concepts2.
- Concepts.keep_most_specific() : garde uniquement les concepts les plus spécifiques, c'est à dire que l'on enlève tous les concepts plus généraux qu'un autre.
- Concepts.keep_most_generic() : garde uniquement les concepts les plus généraux, c'est à dire que l'on enlève tous les concepts plus spécifiques qu'un autre.
- Concepts.lowest_common_ancestors() : retourne les plus petits parents communs à l'ensemble des concepts.
- Concepts.all_subsets() : retourne tous les sous-ensembles inclus dans l'ensemble.

10 Utiliser PyMedTermino sans Python

PyMedTermino peut aussi être utilisé sans Python, simplement pour convertir les données XML de la SNOMED CT et de la CIM10 en bases de données. Les bases de données SQLite3 ainsi créées peuvent ensuite être interrogé avec la plupart des langages de programmation, cependant vous n'aurez pas accès aux fonctions de plus haut niveau proposées par PyMedTermino (comme les fonctions ancestors() et descendants() par exemple).

La définition des tables des bases de données peut être consultée dans les fichiers scripts/import_sonmedct.py et scripts/import_icd10.py.