TD4._sujet

January 27, 2023

1 TD4 Tuple, NLTK - Analyse de textes

Bienvenus au TP4 !

Dans ce TP, on va finir la révision avec un autre data structure dans Python : `tuple`. Vous avez un exercise pour cette première partie.

On va ensuite travailler avec le package NLTK pour la partie l'analyse du texte.

1.1 Partie 1. Another important data structure: Tuples

Tuples are another very frequently used data structure in python. Unlike lists, they are *immutable* ordered sequences.

As they are ordered sequences, you can use slicing, indexing and unpacking with tuples. As they are immutable, you can also use (some) tuples as dictionary keys.

Tuples are omnipresent in python code. Every time you return multiple values at once, you're in fact returning a tuple and unpacking it down the line:

```
def chimp_life(peanuts):
    ...
    # this is strictly equivalent to:
    # return (chimp_action, remaining_peanuts)
    return chimp_action, remaining_peanuts

# this is actually just unpacking!
chimp_action, peanuts = chimp_life(peanuts)
```

In fact, any value followed by a comma is interpreted as a tuple:

```
>>> chimp = 'chimp'
>>> type(chimp)
<class 'str'>
>>> chimp = 'chimp',
>>> type(chimp)
<class 'tuple'>
```

You may also add parentheses around a tuple:

```
>>> monkeys_a = 'chimps', 'mandrill'
>>> monkeys_b = 'chimps', 'mandrill',
>>> monkeys_c = ('chimps', 'mandrill')
```

```
>>> monkeys_d = ('chimps', 'mandrill',)
>>> monkeys_a == monkeys_b == monkeys_c == monkeys_d
True
```

Parentheses are required when you create tuples of tuples, or when creating an empty tuple

```
>>> monkeys_e = ('chimps', 'mandrill'),
>>> monkeys_a == monkeys_e
False
>>> empty = ()
>>> type(empty)
<class 'tuple'>
```

1.1.1 Exercise 1: GCD

Write a function to compute the GCD of two positive integers. You can freely use the fact that `gcd(a, b)` is mathematically equal to `gcd(b, a % b)`, and that `gcd(a, 0) == a`.

If it helps, start by assuming that `a >= b` if you'd like, but your final function should be able to handle all cases, including when `a < b`.

It is possible to accomplish this in three lines of Python code (or with extra cleverness, even fewer!). Consider exploiting tuple packing and unpacking!

Note: The standard library has a `gcd` function. Avoid simply importing that function and using it here - the goal is to practice with tuple packing and unpacking!

[ ]:
```
# Write your code here
def gcd(a, b):
    """Compute the GCD of two positive integers."""
    pass

# testing code
print(
   gcd(10, 25), # => 5
   gcd(25, 10), # => 5

   gcd(14, 15), # => 1
   gcd(15, 14), # => 1

   gcd(3, 9), # => 3
   gcd(9, 3), # => 3

   gcd(1, 1), # => 1
)
```

1.2 Partie 2 - Analyse de textes avec NLTK

[ ]:
```
# Télécharger les librairies et tester qu'elles fonctionnent

import nltk # https://www.nltk.org/install.html
import numpy # https://www.scipy.org/install.html
import matplotlib.pyplot # https://matplotlib.org/downloads.html
import unidecode # https://pypi.python.org/pypi/Unidecode
```

Si vous n'avez pas les librairies, il faut les installer. Si vous n'en avez pas besoin aujourd'hui, ce sera pour plus tard. Faites

`pip install le_nom_de_la_librairie`

1.2.1 Exo2 Créer une variable string

Construire une variable `my_text` combinant les 3 extraits de DinG. faite en sorte que le résultat ne contienne **que le texte**, et non les indexations ni le nom du locuteur.

Votre résultat doit être un string qui ressemble à cela :

```
my_text = "Euh bon après de toute façon on va à la limite on va faire 1 tour
... on va distribuer les ressources qui correspondent et ensuite on va euh on
va faire les différentes actions"
```

[ ]:
```
ding1 = "0091 R   Euh bon après de toute façon on va à la limite on va faire 1␣
 ↪tour pour euh alors oui si par contre au départ on va devoir à tour de rôle␣
 ↪placer notre première ville de départ plus 1 route euh adjacente à notre␣
 ↪ville"
ding2 = "0119 R   Euh faut que tu places aussi une route dans la direction qui␣
 ↪t'intéresse euh alors tout ce qui est construit normalement ne peut plus␣
 ↪être euh déconstruit ouais on peut pas détruire quelque chose par contre␣
 ↪ouais il faut peut-être qu'on attende un tout petit peu parce que c'est on␣
 ↪peut considérer que c'est le début du jeu"
ding3 = "0133 R   Alors ouais pour bien faire donc là au le premier tour de jeu␣
 ↪ça va être vraiment ça ça va être on va chacun mettre 2 moins on met une␣
 ↪ville chacun puis une deuxième ville avec une route adjacente et ensuite à␣
 ↪chaque tour on commence par lancer les dés ensuite on peut faire du commerce␣
 ↪euh et ensuite on fait nos achats et quand on a fini bah c'est le suivant␣
 ↪qui va lancer les dés et euh les les on va distribuer les ressources qui␣
 ↪correspondent et ensuite on va euh on va faire les différentes actions"
```

[ ]:
```
# Mettre votre code ci-dessous

my_text = #TODO

print(my_text)
```

1.2.2 Exo3 Tokenizer une chaîne de caractères

- Utiliser `nltk.word_tokenize` pour tokeniser.
- Écrire une fonction qui prend en paramètre une chaine de caractère, applique le tokenizer et assigne le résultat dans une vairable `tokens`.
- Remplacer toutes les majuscules avec les minuscules. Considérer la function `.lower()`
- Afficher (print) les 10 premiers tokens et retourner (return) la liste de tous les tokens.

Les 10 premiers tokens à afficher sont :

['euh', 'bon', 'après', 'de', 'toute', 'façon', 'on', 'va', 'à', 'la']

[ ]:
```
from nltk.tokenize import word_tokenize
from nltk.text import Text
```

[ ]:
```
# TODO
def tokenify(my_str):
    pass

text_tokens = tokenify(my_text)
```

1.2.3 Exo4 - Exploration de la chaîne de caractères - objets `Text`

Maintenant nous allons travailler avec l'objet `Text`. Les prochaines étapes vous permettent de vous familiser avec des functions de cet objet.

(1) Ecrire une fonction qui transforme les tokens en objet `nltk.text.Text`, cela permettra d'accéder aux méthodes de la classe text dans le package NLTK.

Une fois que les tokens sont transformés en objct Text, vous pouvez vérifier en printant `print(type(t))`, et il va s'afficher :

```
<class 'nltk.text.Text'>
```

[ ]:
```
# TODO

def texify (tokens):
    pass

t = texify(text_tokens)
print(type(t))
```

(2) Dans quel contexte le mot `jeu` et `dés` appariassent-ils dans le texte `t` ?

Pour cela, vous pouvez utiliser la fonction `t.concordance(mot_en_question)` de l'object `Text`. Cette fonction vous donne les mots avant et après de votre mot-clé.

Afficher le résultat.

Normalement vous aurez 2 matches pour le mot 'jeu' et 2 matches pour le mot 'dés'.

[ ]:
```
# TODO
```

(3) Quels autres mots apparaissent dans un contexte similaire ? Nous pouvons le découvrir avec la fonction `similaire()`. Vous mettez le mot en question entre parenthèses, tout comme la syntaxe de `concordance()`:

Quel est le mot similaire de "lancer" ? Afficher le résultat.

[ ]:
```
# TODO
```

(4) Pour faciliter le traitement des mots, nous les indexez avec des chiffres.

L'object `Text` possède une function qui peut utomatiquement indexer un vocabulaire.

Indexer le mot "jeu" avec la méthode `.index()`.

Afficher le résultat. Vous avez quoi comme chiffre pour ce mot ?

Vous pouvez également accéder aux mots avec leur indices :

```
print(t[:10])
```

```
['euh', 'bon', 'après', 'de', 'toute', 'façon', 'on', 'va', 'à', 'la']
```

[ ]:
```
# TODO
```

(5) Pour trouver la location des mots dans un texte, nous pouvons l'afficher à l'aide d'un diagramme de dispersion.

Essayer la function `dispersion_plot([mot1, mot2, ...])`, afficher les positions des mots "jeu", "dés", "ville".

[ ]:
```
# TODO
```

(6) Plot de la distribution des fréquences des mots: tout simplement utiliser la fonction `plot()`. Dans les parenthèses, vous pouvez indiquer le nombre des mots que vous voulez regarder.

Afficher le plot les 20 mots les plus fréquents dans `t` avec `plot`

[ ]:
```
# TODO
```

1.2.4 /! Submission instructions

You will need to submit this lab on Arche before 9:59am on **Friday, 10th February**. Submit either a `.py` or an `.ipynb` file containing the functions you wrote for all the exercises and name it `td4_firstname_lastname.py` or `td4_firstname_lastname.ipynb` accordingly, where `firstname` should be your first name and `lastname` should be your last name (e.g. Jane Doe's submission should be called `td4_jane_doe.py` or `td4_jane_doe.ipynb`).