Formation Python Module 5

Ihab ABADI - UTOPIOS

Introduction Test

Ihab ABADI - UTOPIOS

Test et clean Test

- Le génie logiciel est une question de savoir-faire où nous devons écrire toutes les parties du logiciel avec le même soin, qu'il s'agisse de code de production ou de test.
- Rédiger des tests fait partie de notre savoir-faire.
- Nous ne pouvons avoir un code propre que si nous avons des tests propres.
- Un test propre se lit comme une histoire.
- Un test propre doit contenir toutes les informations nécessaires pour comprendre ce qui est testé.

Clean Test

- Imaginons que nous construisons une application de boutique en ligne. Les utilisateurs peuvent rechercher des produits, les sélectionner, collecter les produits dans un panier et enfin les acheter. Dans le cadre de l'application, nous avons un cas d'utilisation avec les spécifications suivantes :
- Étant donné un panier contenant des produits d'un total de 50 000 => Given
- Lorsque le total est calculé => When
- Ensuite, il renvoie 50 en tant que total => Then

Ecriture d’un test clean

- Le nom d'un test doit révéler le cas de test exact, y compris le système testé.
- Il doit spécifier l'exigence du cas de test aussi précisément que possible.
- L'objectif principal d'un bon nom de test est que si un test échoue, nous devrions être en mesure de récupérer la fonctionnalité cassée à partir du nom du test.
- Deux conventions de dénomination populaires :
  - **GivenWhenThen** (ex : GivenUserIsNotLoggedIn_whenUserLogsIn_thenUserIsLoggedInSuccessfully)
  - **ShouldWhen** (ex : ShouldHaveUserLoggedIn_whenUserLogsIn)

Ecriture d’un test clean

- L'utilisation de noms propres, significatifs et révélateurs d'intention dans les tests est aussi important que l'utilisation de noms propres dans le code de production. Par conséquent, nous devrions utiliser des dénominations propres dans des domaines tels que :
  - noms d'éléments logiciels (par exemple : noms de classe, de fonction, de variable)
  - scénarios de préparation
  - exécution du système testé
  - affirmations des comportements attendus
- Conseils pour nommer vos éléments logiciels :
  - suivre les conventions de nommage
  - ne polluez pas les noms avec des détails techniques
  - utiliser des dénominations fonctionnelles relatives au domaine métier
  - utiliser des constantes nommées pour les nombres/chaînes magiques
  - utiliser des noms prononçables
  - ne pas utiliser d'abréviations personnalisées
  - être explicite plutôt qu'implicite
  - révéler l'intention avec des fonctions bien nommées au lieu d'utiliser des commentaires

Ecriture d’un test clean - AAA

- Le modèle Arrange-Act-Assert est une manière descriptive et révélatrice d'intention de structurer des cas de test. Il prescrit un ordre des opérations:
  - La section Arrange doit contenir la logique de configuration des tests. Ici, les objets sont initialisés et préparés pour l'exécution des tests.
  - La section Act invoque le système que nous sommes sur le point de tester. Il peut s'agir par exemple d'appeler une fonction, d'appeler une API REST ou d'interagir avec certains composants.
  - La section Assert vérifie que l'action du test se comporte comme prévu. Par exemple, nous vérifions ici la valeur de retour d'une méthode, l'état final du test , les méthodes que le test a appelées, ou les éventuelles exceptions attendues et les résultats d'erreur.

Ecriture d’un test clean – F I R S T

- F.I.R.S.T est un acronyme contenant 5 caractéristiques importantes d'un test propre.
- **Fast**
- **Independent**
- **Repeatable**
- **Self-validating**
- **Thorough**

Ecriture d’un test clean

- Un test doit vérifier un seul comportement. Un même comportement peut contenir une ou plusieurs lignes d'assertions dans le code. Un test doit être couplé à un comportement fonctionnel et non à une action technique ou à une modification du code.
- Utiliser des données de test significatives
  - Les tests sont des exemples d'utilisations de code. Ils doivent utiliser des données de test significatives relatives au domaine de l'entreprise, résultant en des exemples lisibles, utilisables et réels. Par conséquent, révéler la connaissance du domaine en utilisant des données de test significatives est essentiel pour produire des tests propres.
- Masquer les données non pertinentes pour le test
  - Ne polluez pas vos tests avec des données de test non pertinentes. De telles informations ne font qu'augmenter la charge mentale cognitive, ce qui entraîne des tests gonflés. Au lieu de cela, masquez les données non pertinentes en utilisant des générateurs de données de test.

Introduction à PyTest

Ihab ABADI - UTOPIOS

Qu'est-ce que PyTest ?

- Framework de test utilisé pour écrire et exécuter des tests
- Principalement utilisé dans le cadre de la réalisation d'API REST
- Utile dans beaucoup de situations (tests d'applications, BDD, …)
- Avantages:
  - Possible de lancer plusieurs tests en parallèle (durée d'exécution réduite)
  - Détection automatique des fichiers de test
  - Notion de sous-sections
  - Gratuit et open-source
  - Simple d'utilisation, syntaxe intuitive, documentation docs.pytest.org
- Installer via la commande

```
pip install pytest
```

Les fichiers de test

- Lancements des tests avec la commande `pytest` dans le répertoire des fichiers de test
- Par défaut, recherche des fichiers en **test_*.py** ou ***_test.py**
- Choisir un fichier `pytest <filename>`
- Choisir un test `pytest <filename>::<test_func>`
- /!\ Les noms des fonctions de test doivent commencer par **« test* »** les autres ne seront pas exécutées
- Démo

Options de lancements utiles

- Possible d'augmenter ou diminuer la clarté des retours avec `-v` ou `-q` (verbosité)
- `--collect-only` pour collecter les tests sélectionnés sans exécuter
- Pour arrêter l'exécution avec un nombre maximum de tests échoués: `pytest –maxfail=<number>`
- Changer le répertoire de base : `pytest --rootdir=<folder>`
- Ignorer un chemin : `pytest --ignore=<filepath>`
- Pour plus d'infos `--help` ou utiliser la doc

Options de lancements utiles

- Il est aussi possible de lancer pytest depuis du code python directement via cette syntaxe :

```
retcode = pytest.main(["-x", "mytestdir"])
```

- On passera ici les arguments dans la liste en paramètre et le retour correspondra au exit code de pytest

Exercice

EXERCICE

- En utilisant pytest et les méthodes définies dans l'api task.py
  - Créer une ou plusieurs fonctions test pour la méthode add.
  - Créer une ou plusieurs fonctions test pour s’assurer que l’id de chaque task est unique.

Sous-Sections : mots-clés et marqueurs

- Problèmes d'organisation dans des plus gros projets, test nombreux, besoin d'organisation → Importance des sous-sections
- Mots-clés :
  - En se basant sur le nom des tests (des fonctions)
  - Commande `pytest -k <keyword>`
- Marqueurs :
  - Définir les marqueurs dans un fichier `pytest.ini`
  - Utiliser le décorateur `@pytest.mark.<markername>`
  - Commande `pytest -m <markername>`
- Possible d'utiliser les string expressions (not, and, or)
- Démo

Exercice

EXERCICE

- Créer un marqueur smoke .
- Ajouter ce marqueur à certaine fonction de test de la fonction add de l’exercice 1
- Exécuter uniquement les smoke test.

Marqueur parametrize

- Possible d'exécuter plusieurs fois un même test avec différents paramètres passés via une liste de tuples
- Décorateur :

```
@pytest.mark.parametrize("p1, p2", [(1,2),(2,4)])
def test_double(p1, p2):
    assert p1*2 == p2
```

- Démo

Exercice

EXERCICE

- Créer une fonction de test qui permet de tester l'ajout d'une task et la récupération de la task et qui s'exécute avec différentes tasks.

Marqueur xfail

- Dans le cas où un test est censé toujours échouer, il est possible d'utiliser le marqueur xfail `@pytest.mark.xfail`
- Lors de l'exécution, il sera noté « XFAIL »
- S'il réussit, pytest le noteras comme « XPASS »
- Il est possible de considérer XPASS comme un FAIL en passant l'argument `strict=True`
- Possible de préciser la raison avec `reason=<string>`
- Démo

Exercice

EXERCICE

- Ajouter un marqueur pour faire échouer ce test.

```
def test_unique_id_is_a_duck():
        uid = tasks.unique_id()
        assert uid == 'a duck'
```

- Ajouter un marqueur pour faire échouer ce test pour les version de l'application task < 1.2

```
def test_unique_id_1():
        id_1 = tasks.unique_id()
        id_2 = tasks.unique_id()
        assert id_1 != id_2
```

Marqueur skip

- Possible de passer (skip) certains test grâce avec des markers dédiés
- Skip:
  `@pytest.mark.skip`
- Skip conditionnel:
  `@pytest.mark.skipif(<condition>)`
- Possible de préciser la raison avec `reason=<string>`
- Démo

Exercice

EXERCICE

- Suite à une erreur dans la compréhension de l'api task par l'équipe dev, on souhaite ne pas exécuter le test suivant:
- Ajouter un marqueur pour passer l'exécution de ce test, ainsi qu'un message.

```
def test_unique_id_1():
        id_1 = tasks.unique_id()
        id_2 = tasks.unique_id()
        assert id_1 != id_2
```

Classes de Test

- PyTest permet l'utilisation de test par l'intermédiaires de classes
- Pour ce faire, ils faut que la classe soit nommée **« Test* »** et que ses méthodes soient, comme les fonctions classiques, nommées **« test* »** (ne pas oublier l'attribut self)
- Cela peut être utile pour :
  - Regrouper les tests par classes pour l'organisation
  - Partager les fixtures uniquement à la classe
  - Appliquer des marqueurs au niveau de la classe (ce qui impacte tout les tests)
- /!\ Chaque test utilise sa propre instance de la classe, mais il reste possible d'utiliser les attributs de classe entre les tests
- Démo

Exercice

EXERCICE

- En utilisant pytest et les méthodes définies dans l'api task.py
  - Créer une classe test pour regrouper les tests de la fonction update.
  - Marquer cette classe en smoke.
  - Exécuter cette classe de tests.

Tests d'Exceptions

- Si l'on veut vérifier si une exception est levée dans un test, pytest fournit un contexte dédié

```
with pytest.raises(<exception>):
    <code censé lever l'exception>
```

- Démo

Exercice

EXERCICE

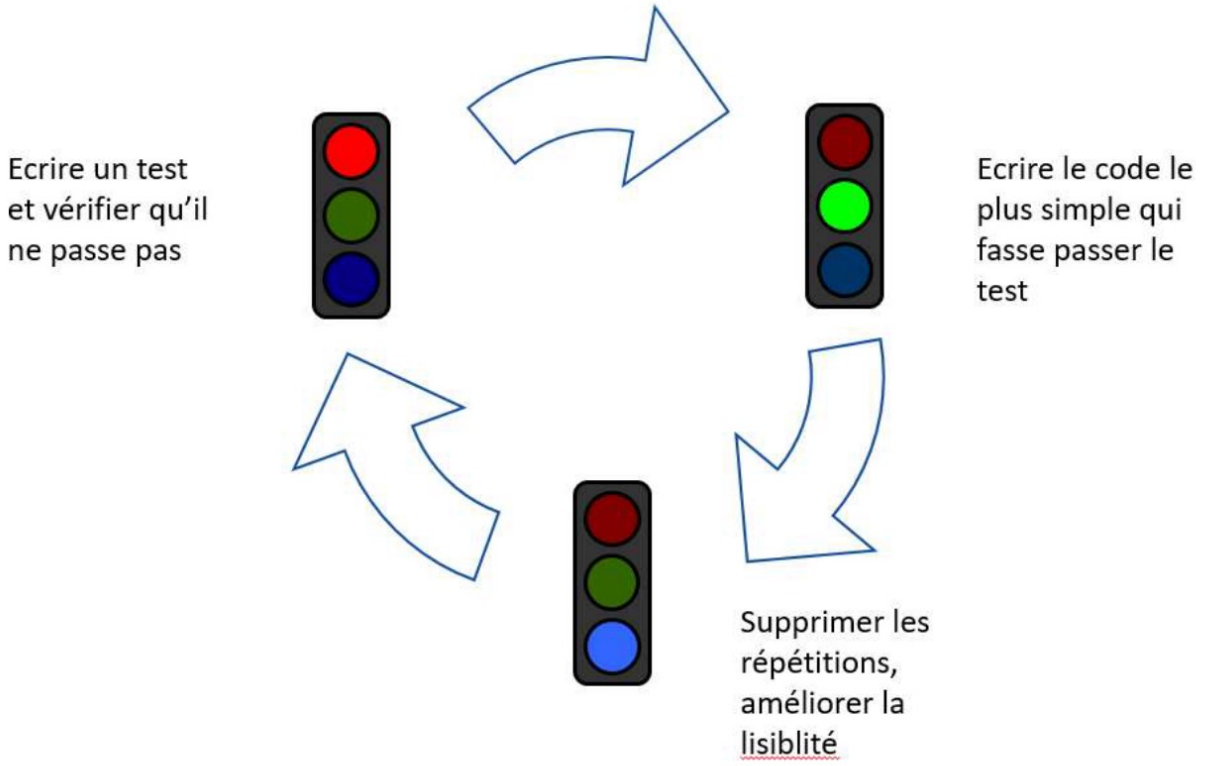
- Créer une classe test pour tester les exceptions levées par les méthodes add, update, list et get du module tasks
- Marquer les tests des méthodes get et list en smoke.
- Exécuter les tests.

TDD

TDD

- C’est une technique de développement
- Les tests aident à spécifier du code, et non pas à le valider
- 3 étapes pour développer une fonctionnalité ou corriger un bug
  - Je pose un test unitaire et je le fais passer au rouge
  - J’écris le code nécessaire pour faire passer mon test au vert
  - Je nettoie mon code et le refactorise
- La couverture de code devient alors un bénéfice, et non plus un objectif
- Le vrai indicateur à regarder est : la NON-couverture de code

TDD


Ecrire un test
et vérifier qu'il
ne passe pas
Ecrire le code le
plus simple qui
fasse passer le
test
Supprimer les
répétitions,
améliorer la
lisiblité

TDD

- TDD renverse le modèle classique
  - Besoins -> Spécifications -> Codage/Tests Unitaires -> Tests d'intégration -> Maintenance
  - Scénarios Utilisateurs -> Test/Code/Refactor -> Tests d'Intégration
- TDD remplace une approche « industrielle »...
  - Concevoir, puis produire, puis valider, puis analyser, puis corriger
  - Rationnaliser la production de code : le mieux est l'ennemi du bien
- ...par une approche « artisanale »
  - Le code est un matériau de production, on cherche l'excellence dans le geste
  - Tester et maintenir le code au plus près de son écriture
  - Principes KISS et SOLID (Clean code)

TDD – quelques principes de Clean code

- KISS : Keep It Simple Stupid
- SOLID
  - Single Responsibility Principe : une classe doit avoir une seule responsabilité
  - Open/Closed : une classe doit être ouvert à l'extension mais fermée à la modification
  - Liskov Substitution : utilisez le type le plus « haut » possible (classes abstraites, interfaces...)
  - Interface segregation : préférez utiliser plusieurs interfaces spécifiques pour chaque client plutôt
    qu'une interface générale
  - Dependency Inversion : injection de dépendances. Il faut dépendre des abstractions et non pas des
    implémentations
- DRY : Don't Repeat Yourself

TDD - Récap

- Commencez toujours par un test automatisé qui échoue
- N’ajoutez jamais de tests sur la barre rouge
- Eliminez toute duplication

Exercice

EXERCICE

- En appliquant les principes du TDD et pytest.
- Écrivez une méthode "fizzBuzz" qui accepte un nombre en entrée et renvoie une chaîne de caractère.
  - Pour les multiples de trois, retournez "Fizz" au lieu du nombre
  - Pour les multiples de cinq, retournez "Buzz"
  - Pour les nombres multiples de trois et de cinq, renvoyez « FizzBuzz ».
  - Pour les autres cas, elle renvoie le nombre
- Remarques:
  - commencer par écrire les tests nécessaire et les mettre en échec.
  - Coder la méthode pour répondre au tests
  - Refactoriser votre code.

Exercice

EXERCICE

- En appliquant les principes du TDD et pytest.

Implémenter une fonctionnalité de recherche de ville. La fonction prend une chaîne (texte de recherche) en entrée et renvoie les villes trouvées qui correspondent au texte de recherche.

Exemple de villes : Paris, Budapest, Skopje, Rotterdam, Valence, Vancouver, Amsterdam, Vienne, Sydney, New York, Londres, Bangkok, Hong Kong, Dubaï, Rome, Istanbul

Conditions:

1. Si le texte de la recherche contient moins de 2 caractères, aucun résultat ne devrait être renvoyé.

2. Si le texte de recherche est égal ou supérieur à 2 caractères, il doit renvoyer tous les noms de ville commençant par le texte de recherche exact.

   Par exemple, pour le texte de recherche "Va", la fonction doit renvoyer Valence et Vancouver

3. La fonctionnalité de recherche doit être insensible à la casse

4. La fonctionnalité de recherche devrait également fonctionner lorsque le texte de recherche n'est qu'une partie d'un nom de ville

   Par exemple "ape" devrait renvoyer la ville "Budapest"

5. Si le texte de recherche est un « * » (astérisque), il doit renvoyer tous les noms de ville.

Fixtures

Les Bases

- Fonctions qui s'exécutent avant chaque test
- Utilisées pour initialiser les données de test
- Permets d'éviter les répétitions (fichiers, BDD, …)
- Utilisation du décorateur `@pytest.fixture`
- La fonction porte le nom du paramètre utilisé en entrée du test et permet de l'initialiser, on peut changer le nom avec l'attribut `name=<string>`

- Démo

Exercice

EXERCICE

- Créer une fixture qui permet de générer plusieurs tasks(3 par exemple).
- Créer un test pour l'ajout des tasks générées par la fixture.
- Exécuter le test.

Scopes

- Définit la portée de la fonction et le nombre d'exécution lors d'une série de tests
- Types de scopes:
  - Function (défaut): Fixture détruite à la fin du test de la fonction
  - Class : Fixture détruite à la fin du test de la classe
  - Module : Fixture détruite à la fin du test du module python
  - Package : Fixture détruite à la fin du test du package python
  - Session : Fixture détruite à la fin de la session de tests

Exercice

EXERCICE

- Faire en sorte que la fixture de l’exercice slide 32 soit exécuter pour la totalité de la session de test en ayant au moins 2 tests.
- Exécuter les tests en montrant le nombre d’exécution de la fixture.

Fixture autouse

- Il est possible d'avoir une fixture en autouse, cette fixture sera appelée automatiquement avant chaque tests et cela sans qu'elle soit passée en paramètre de fonction
- Pour passer une fixture en autouse, il faudra ajouter le paramètre au décorateur `autouse=True`
- Cette fixture se verra automatiquement utilisée si :
  - Les tests se trouvent dans le même module que la fixture
  - Les tests se trouvent dans la même classe que la fixture

Exercice

EXERCICE

- Créer une fixture qui permet d'initier la base de données du module task. Cette fixture doit être utilisée avec chaque fonctions de tests.
- Vérifiez son utilisation en reprenant les tests effectués plus tôt et n'utilisez plus « initialized_tasks_db() ».

Fixtures multiples

- Si plusieurs fixtures doivent être utilisées :
- Il est possible de passer une première fixture en paramètre d'une autre et d'ensuite appeler l'autre dans un test, exemple :

```
def first_fixture(): …
def second_fixture(first_fixture): …
def test_function(second_fixture): …
```

- Il est aussi possible de passer plusieurs fixtures en paramètres

- Démo

Exercice

EXERCICE

- Créer une fixture qui permet de générer plusieurs tasks(3 par exemple).
- Créer une deuxième fixture qui ajoute les tasks de la première fixture.
- Créer un un test qui récupère la deuxième fixture et qui ajoute une tasks et qui test le nombre de tasks.

Fixtures paramétrées

- Comme pour les tests avec le décorateur parametrize, il est possible d'avoir des fixtures paramétrées
- Chaque test qui utilisera n fixtures paramétrées produira une série de test avec chaque ensemble d'entrées possible (produit cartésien)

- Démo

Exercice

EXERCICE

- Refaire l’exercice slide 40 en récupérant dans la première fixture un paramètre avec 3 tasks.

Fixtures yeild

- Lorsque l'on utilise le mots clé `yield` à la place du `return` dans une fixture, son fonctionnement change
- Toutes les instructions après le `yield` seront exécutées après chaque test utilisant la fixture
- Les instructions après le `yield` seront exécutées quelque soit le résultat des tests

Exercice

EXERCICE

- Ecrire une fixture qui permet à la fois d'initialiser la base de données avant chaque tests et arrêter la base après les tests.
- Utiliser cette fixture avec la classe de tests de la fonction update

Le fichier conftest.py

- Il est possible d'utiliser les mêmes fixtures dans plusieurs fichiers de test
- Pour cela il faut créer un fichier nommé `conftest.py` dans lequel on pourra définir ces fixtures

- Démo

Exercice

EXERCICE

- Regrouper toutes les fixtures nécessaires pour les tests du module tasks dans un seul fichier.

Builtin Fixtures

- Il existe plusieurs Fixtures prédéfinies par pytest, pour les utiliser il suffit de passer leurs noms en paramètre
- Quelque exemple de buitin fixtures:
  - Tmpdir et tmpdir_factory
  - Cache
  - Capsys
  - Monkeypatch
- Démo
- Pour plus d'informations : https://docs.pytest.org/en/7.1.x/builtin.html

Mocks

Définition des mocks

- Les mocks sont des objets qui permettent d'imiter le comportement de librairies ou de modules que nous ne souhaitons pas tester ou que nous ne devons pas tester.
- Pour réaliser les mocks:
  - Nous pouvons utiliser les objets de mock de unitest.
  - Nous pouvons utiliser les fixtures : une combinaison d'une custom fixture et monkeypatch qui est une builtin fixture.
- Démo

Exercice

EXERCICE

- En utilisant l’exercice du slide 17 :
- Créer un mock pour la base de données

Test en Parallèle

Ihab ABADI - UTOPIOS

Test en Parallèle

- Par défault, PyTest va lancer nos tests de façon séquentielle.
- Dans un scénario réel (par exemple dans le cadre d'un projet logiciel client), il y aura énormément de fichiers de tests contenant chacun plusieurs fonctions. Pour accélérer le processus de test, il convient alors de réaliser ces derniers en parallèle.
- Pour ce faire, on peut installer le plugin pytest-xdist via la commande pip
- Ensuite, il suffit simplement de modifier encore une fois notre commande de lancement des tests pour prendre en compte le lancement de ces derniers de façon parallèle. Il faut donc exécuter la commande **pytest –n <number>** où « number » correspond au nombre de threads que l'on veut créer pour cette suite de tests

Exercice

Exercice

Lancer les tests de l’exercice FizzBuzz dans 10 Threads

Module RE

Ihab ABADI - UTOPIOS

Expressions régulières

- Les expressions régulières en Python sont géré par le module re
- Re permet de définir une chaîne qui contient des symboles et des caractères spéciaux pour rechercher et extraire les informations.
- Re permet les opérations suivantes :
  - Rechercher
  - Comparer
  - Extraire
  - Fractionner
- Aussi appelé regex

Les méthodes du modules re sont :

compile()

search()

match()

findall()

split()

…

Expressions régulières – re fonctionnement

- Etape 1:
  - Compiler le pattern avec la méthode compile ou une autre méthode
- Etape 2:
  - Exécuter une méthode du module re
- Etape 3
  - Afficher ou exploiter le résultat.
- Démo

Expressions régulières – Méthode search du module re

- La méthode search permet à la fois de compiler et exécuter la regex.
- Cette méthode renvoie le premier élément trouvé ou false.
- Démo

Expressions régulières – Méthode match du module re

- La méthode match renvoie la chaine de caractère trouvée au début de la chaine.
- La méthode match renvoie faux si rien n’est trouvé
- Démo

Utilisation du search (5min)

EXERCICE

- 1 - Vérifiez si une chaîne de caractère. Contient 0xB0. Affichez le résultat sous forme de booléen.
- 2- Filtrer tous les éléments d’une liste qui ne contiennent pas e.

Expressions régulières – Méthode split du module re

- La méthode split permet d'envoyer une collection de données en fonction du pattern de la regex.
- Démo

Utilisation du split et search (5min)

EXERCICE

- Pour une chaîne d'entrée donnée, affichez toutes les lignes ne contenant pas start, quelle que soit la casse.

Expressions régulières – Méthode sub module re

- La méthode sub permet d'effectuer une recherche en fonction d'un pattern et remplacer le résultat trouvé par une autre chaine.
- Démo

Utilisation du sub(10min)

EXERCICE

- Remplacer toutes les occurrences de 5 par cinq dans une chaine de caractères.
- Remplacer la première occurrence de 5 par cinq dans une chaine de caractères.
- Remplacer, dans une chaine de caractère, toutes les occurrences de la note, quelle que soit la casse, par X

Expressions régulières – Méthode findAll module re

- La méthode findAll permet d'extraire la totalité des éléments correspondant à un pattern.
- Démo

Expressions régulières – séquences des caractères de recherches

Character	Description
\d	Represents any digit(0 - 9)
\D	Represents any non-digit
\s	Represents white space Ex: \t\n\r\f\v
\S	Represents non-white space character
\w	Represents any alphanumeric(A-Z, a-z, 0-9)
\W	Represents non-alphanumeric\b
\b	Represents a space around words
\A	Matches only at start of the string
\Z	Matches only at end of the string

Expressions régulières – séquences des caractères de recherches

character	Description
\b	Matches only one space
\w	Matches any alpha numeric character
{5}	Repetition character

Expressions régulières – Quantifiers

Character	Description
+	1 or more repetitions of the preceding regexp
*	0 or more repetitions of the preceding regexp
?	0 or 1 repetitions of the preceding regexp
{m}	Exactly m occurrences
{m, n}	From m to n. m.defaults to 0 n.defaults to infinity

Expressions régulières – caractères spéciales

Character	Description
\	Escape special character nature
.	Matches any character except new line
^	Matches begining of the string
$	Matches ending of a string
[...]	Denotes a set of possible characters Ex: [6b-d] matches any characters 6, b, c, d
[^...]	Matches every character except the ones inside brackets Ex: [^a-c6] matches any character except a, b, c or 6
(...)	Matches the RE inside the parentheses and the result can be captured
R | S	matches either regex R or regex S

Expressions régulières

- Démo

Utilisation des Ancres (15min)

EXERCICE

- Pour une chaîne d'entrée donnée, remplacez uniquement mot rouge par jaune.
- Pour une liste d'entrée donnée, filtrez tous les éléments qui commencent par per ou se terminent par in.
- Pour une liste d'entrée donnée, remplacez hand par X pour tous les éléments qui commencent par hand suivi d'au moins un caractère de mot.
- Pour une liste d'entrée donnée, filtrez tous les éléments commençant par h. De plus, remplacez e par X pour ces éléments filtrés.

Portion, lambda (15min)

EXERCICE

- Pour une chaîne donnée qui contient : exactement une fois. Extraire tous les caractères après le « : ».
- Remplacez toutes les occurrences de « par » par « spar », « spare » par « extra » et « park » par « garden » pour une liste de chaînes données.
- Extrait tous les mots entre (et) d'une chaîne donnée sous forme de liste.
- Pour une liste de chaînes donnée, changez les éléments en un tuple d'élément d'origine et le nombre d'occurrence de cet élément.
- Soit la chaîne suivante qui a des champs séparés par :. Chaque champ contient quatre lettres majuscules suivies éventuellement de deux chiffres. Ignorez le dernier champ, qui est vide. En utilisant Match.groups et re.finditer trouvez la sortie suivante :
  - Input => TWXA42:JWPA:NTED01:
  - Output => [('TWXA', '42'), ('JWPA', 'NA'), ('NTED', '01')]

Expressions régulières – Utilisation des regex avec des fichiers

- Nous pouvons utiliser les méthodes du module re avec différentes types de source de données.
- Par exemple des fichiers txt.
- Démo:
- Soit un fichier salaries.txt qui continent l'id, nom, prénom, salaire.
- On souhaite extraine l'id et le salaire pour les enregister dans un second fichier
- Démo

Utilisation de fichier (20min)

TP

- Écrire un script qui permet de convertir le fichier a.ini en a.json

Module Pandas

Ihab ABADI - UTOPIOS

Introduction Pandas

- Bibliothèque d'analyse de données Pyton
- Construit sur Numpy
- Abréviation de Panel Data System
- Utilisé en production dans de nombreuses entreprises
- Gestion des données
- Analyse des données
- Modélisation des données
- Organiser les données sous une forme adaptée au tracé ou au tableau

Introduction Pandas

- Pandas fournit deux types de structures de données.
  - Series
  - DataFrames.
- fournit des outils pour la manipulation des données : remodelage, fusion, tri, découpage, agrégation, etc

Pandas – Series

- **Series** est un tableau unidimensionnel étiqueté capable de contenir des données de n'importe quel type (entier, chaîne, flottant, objets python, etc.).
- Les étiquettes des axes sont collectivement appelées index
- Démo Création des series

Pandas – DataFrames

- Un DataFrame est une structure de données bidimensionnelle
- c'est-à-dire que les données sont alignées de façon tabulaire en lignes et en colonnes.
- **Caractéristiques de DataFrame**
  - Les colonnes peuvent être de types différents
  - Taille - Mutable
  - Axes étiquetés (lignes et colonnes)
  - Peut effectuer des opérations arithmétiques sur les lignes et les colonnes.
- Démo

Pandas – DataFrames – Source externes

- Il existe un certain nombre de commandes pandas pour lire les données à partir des sources externes.
- `pd.read_excel('myfile.xlsx',sheet_name='Sheet1' , index_col=None, na_values=['NA'])`
- `pd.read_stata('myfile.dta')`
- `pd.read_hdf('myfile.h5','df’)`

Demo

Pandas – DataFrames – Exploration des données - attributs

- Récupérer les types de la dataframes avec l'attribut dtypes.
- Récupérer le type d'une colonne avec la l'attribut dtype.
- Récupérer la liste des colonnes avec l'attribut columns.
- Récupérer les noms de colonnes et les labels des lignes avec axes.
- ….

Pandas – DataFrames – Exploration des données - Méthodes

- head([n]), tail([n]), pour récupérer les premiers et les derniers n lignes
- describe() pour générer des statistiques descriptives (pour les colonnes numériques uniquement)
- max(), min() pour renvoyer les valeurs max/min pour toutes les colonnes numériques
- mean(), median() pour renvoyer les valeurs moyennes/médianes pour toutes les colonnes numériques
- std() pour renvoyer l'écart-type
- sample([n]) pour renvoyer un échantillon aléatoire de données
- dropna() supprimer tous les enregistrements avec des valeurs manquantes

Exercice – 15 min

EXERCICE

- En utilisant le fichier salaries.csv
- Récupérez les 20 premiers enregistrements.
- Récupérez les 10 derniers enregistrements.
- Trouvez le nombre d'enregistrements de cette dataframe.
- Quels sont les noms de colonnes ?
- Quels types de colonnes avons-nous dans ce bloc de données ?
- Calculer l'écart type pour toutes les colonnes numériques ;
- Quelles sont les valeurs moyennes des 50 premiers enregistrements de l'ensemble de données ?

Pandas – DataFrames – Exploration des données - Sélection

- Pour sélectionner une colonne, nous pouvons :
  - Méthode 1 : sous-ensemble du bloc de données à l'aide du nom de la colonne : df[‘col_name']
  - Méthode 2 : utilisez le nom de la colonne comme attribut : df.col_name

Pandas – DataFrames – Exploration des données – Méthode groupby

- En utilisant la méthode "group by", nous pouvons :
  - Diviser les données en groupes en fonction de certains critères
  - Calculer des statistiques (ou appliquer une fonction) à chaque groupe
- Une fois l'objet groupby créé, nous pouvons calculer diverses statistiques pour chaque groupe
- Aucun regroupement / fractionnement ne se produit tant que cela n'est pas nécessaire. La création de l'objet groupby vérifie uniquement que vous avez passé un mappage valide
- Par défaut les clés de groupe sont triées lors de l'opération groupby. Vous pouvez utiliser le paramètre sort=False pour une accélération potentielle
- Démo

Pandas – DataFrames – Exploration des données – Méthode de filtre

- Pour sous-ensemble les données, nous pouvons appliquer l'indexation booléenne.
- Cette indexation est communément appelée filtre.
- Par exemple, si nous voulons sous-ensemble les lignes dans lesquelles la valeur du salaire est supérieure à 120 000 $
- Tout opérateur booléen peut être utilisé pour filtrer un sous-ensemble les données
- Démo

Pandas – DataFrames – Exploration des données – Slicing

- Il existe plusieurs façons de créer un sous-ensemble du bloc de données :
  - une ou plusieurs colonnes
  - une ou plusieurs rangées
  - un sous-ensemble de lignes et de colonnes
- Les lignes et les colonnes peuvent être sélectionnées par leur position ou leur étiquette
- Lors de la sélection d'une colonne, il est possible d'utiliser un seul ensemble de crochets, mais l'objet résultant sera une série (pas un DataFrame)
- Lorsque nous devons sélectionner plus d'une colonne et/ou faire en sorte que la sortie soit un DataFrame, nous devons utiliser des doubles crochets
- Démo

Pandas – DataFrames – Exploration des données – Selection de lignes

- Si nous devons sélectionner une plage de lignes, nous pouvons spécifier la plage en utilisant ": »
- Si nous devons sélectionner une plage de lignes, en utilisant leurs étiquettes, nous pouvons utiliser la méthode loc
- Si nous devons sélectionner une plage de lignes et/ou de colonnes, en utilisant leurs positions, nous pouvons utiliser la méthode iloc
- Démo

Pandas – DataFrames – Exploration des données – Sorting

- Nous pouvons trier les données par une valeur dans la colonne.
- Par défaut, le tri se fera par ordre croissant et un nouveau bloc de données est renvoyé.
- Nous pouvons trier les données en utilisant 2 colonnes ou plus

Pandas – DataFrames – Exploration des données – Valeur NaN

- Les valeurs manquantes sont marquées comme NaN
- Lors de la somme des données, les valeurs manquantes seront traitées comme zéro
- Si toutes les valeurs sont manquantes, la somme sera égale à NaN
- Les méthodes cumsum() et cumprod() ignorent les valeurs manquantes mais les conservent dans les tableaux résultants
- Les valeurs manquantes dans la méthode GroupBy sont exclues
- De nombreuses méthodes de statistiques descriptives ont une option skipna pour contrôler si les données manquantes doivent être exclues. Cette valeur est définie sur True par défaut

Pandas – DataFrames – Exploration des données – Fonctions d'agrégation

Fonctions d'agrégation courantes sont :

- min max
- compte, somme, prod
- moyen, médian, mode, fou
- std, var
- Démo

Exercice – 10 min

EXERCICE

- En utilisant le fichier salaries.csv
- Calculez les statistiques de base pour la colonne des salaires.
- Trouvez combien de valeurs dans la colonne salaire.
- Calculez le salaire moyen.
- Calculez le salaire moyen pour chaque rang

TP Pandas Covid – 60 min

TP

En utilisant le fichier covid_19.csv

1. Écrivez un programme Python pour afficher les 5 premières lignes de l'ensemble de données COVID-19. Imprimez également les informations du jeu de données et vérifiez les valeurs manquantes.

2. Écrivez un programme Python pour obtenir le dernier nombre de cas confirmés, de décès, récupérés et actifs de nouveau coronavirus (COVID-19) par pays.

3. Écrivez un programme Python pour obtenir le dernier nombre de décès confirmés et de personnes récupérées de cas de nouveau coronavirus (COVID-19) par pays/région - province/État.

4. Écrivez un programme Python pour obtenir la province chinoise des cas confirmés, des décès et des cas récupérés de nouveau coronavirus (COVID-19).

5. Écrivez un programme Python pour obtenir les derniers cas de décès par pays du nouveau coronavirus (COVID-19).

6. Écrivez un programme Python pour répertorier les pays où aucun cas de nouveau coronavirus (COVID-19) n'a été récupéré.

7. Écrivez un programme Python pour répertorier les pays dans lesquels tous les cas de nouveau coronavirus (COVID-19) sont décédés.

8. Écrivez un programme Python pour répertorier les pays dans lesquels tous les cas de nouveau coronavirus (COVID-19) ont été récupérés.

9. Écrivez un programme Python pour obtenir les données des 10 principaux pays (dernière mise à jour, pays/région, confirmés, décès, récupérés) du nouveau coronavirus (COVID-19).