TD2 : exceptions / accès aux fichiers / expressions régulières

Exercice 1 : une exception ?

Soit l'algorithme suivant :

```
fonction divise1(a : réel, b : réel) : réel
      c = a / b
      return c

main:
      saisir a
      saisir b
      c = divise1(a,b)
      affiche la division de {a} par {b} donne c
```

1. Coder la fonction **divise1** qui permet de diviser deux nombres passés en argument (a et b) et qui retourne le résultat.
2. Dans le main, l'utilisateur devra saisir deux nombres et appeler la fonction divise1 et afficher finalement le résultat sous la forme :
   `La division de 12.0 par 12.0 donne 1.0`
3. Lorsque vous lancez le programme, vous obtenez une erreur
   `TypeError: unsupported operand type(s) for /: 'str' and 'str'`
   `TypeError` est une exception. Vous pourriez traiter celle-ci à l'aide de `try/except/else` mais cela n'a pas beaucoup de sens dans ce cas, pourquoi ? Ajoutez les modifications nécessaires pour éviter cette exception
4. Faites un test et tout devrait fonctionner. Bravo !
5. Vous allez provoquer une erreur mathématique (a / b) en saisissant pour b une valeur impossible. Notez en commentaire l'exception obtenue (qui se termine par Error).

Nous allons résoudre ce problème de deux façons différentes.

6. En faisant appel à vos connaissances en algorithmie, coder une nouvelle fonction **divise2** en utilisant les conditions - sans supprimer divise1 - permettant d'éviter cette erreur. Vérifier que tout fonctionne et que vous évitez l'erreur en affichant un message à l'utilisateur.

   Exemple d'affichage :

   ```
   Saisir a : 12
   Saisir b : 0
   Erreur, b est égale à 0 et on ne peut pas diviser par zéro
   La division n'est pas possible dans ce cas
   ```

7. Dans une nouvelle fonction **divise3**, vous allez résoudre le même problème que précédemment en utilisant la gestion des exceptions **try/except/else.** A votre avis, quelle solution est à adopter dans ce cas-là, le if ou le try/except/else ?

8. Vous allez relancer votre programme avec soit le divise2 soit le divise3 en fonction de ce que vous aurez discuté avec l'enseignant et au lieu de saisir des nombres, vous allez saisir "aaa". Que se passe-t-il ? Comment résoudre le problème ? Résolvez le problème avec un try/except/else.

9. Pour terminer, vous allez ajouter une boucle (laquelle ? tant que ou for ?) qui permet de saisir les nombres jusqu'à ce que a et b soient des nombres réels.

Exercice 2 : lecture et écriture d'un fichier texte

Avec Python, les fonctions à utiliser pour manipuler les fichiers sont : `open, read, write, close`.

- ✓ Pour ouvrir un fichier, la syntaxe est : `fichier = open(nom_du_fichier, mode)`, les modes sont :
  - **r**, pour une ouverture en lecture (READ).
  - **w**, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas, Python le crée.
  - **a**, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas, Python le crée.
  - **b**, pour une ouverture en mode binaire.
  - **t**, pour une ouverture en mode texte.
  - **x**, crée un nouveau fichier et l'ouvre pour écriture
- ✓ Pour fermer un fichier : `fichier.close()`. C'est une opération nécessaire, vous pouvez perdre des données (mécanisme de cache) si vous ne faites pas l'opération correctement (vous pouvez l'assimiler à arracher une clé USB)
- ✓ Pour lire le fichier dans sa totalité : `fichier.read()`
- ✓ Pour lire le fichier ligne par ligne : `fichier.readline()` ou si vous voulez lire le fichier ligne par ligne totalement, une boucle permettra de faire l'opération comme dans le code suivant :

```
for ligne in fichier:
    ligne = ligne.rstrip("\r\n") # A quoi sert rstrip ???
    print(ligne)
```

- ✓ Pour écrire dans un fichier (voir le mode nécessaire) : `fichier.write("coucou\n")`.

Travail à faire :

1. Créer un fichier texte avec du contenu que vous pourrez trouver (par exemple : https://fr.wikipedia.org/wiki/Python_(langage)) et lisez le fichier en affichant ligne par ligne le contenu.
2. En reprenant le code, vous allez écrire chaque ligne dans un nouveau fichier (vous devrez pour cela intercaler le code d'écriture dans le code de lecture). Vous faites en sorte que les chaînes vide ne soient pas prises en considération.

Une deuxième façon de faire pour manipuler les fichiers avec Python est d'utiliser la commande **with** qui permet d'assurer que le fichier est fermé à la sortie du **with**. C'est la méthode la plus utilisée en

Python et ce que nous vous conseillons d'utiliser puisque la gestion de la fermeture (surtout en cas d'erreur) sera assurée par la fonction with. Voici un exemple :

```
with open(nom, "r") as infile:
    # Lecture du fichier
    ligne = infile.read()
    # Vous pouvez également utiliser la lecture ligne par ligne
    # décrit dans l'exercice précédent (for ligne in infile …)
```

3. Reprenez vos codes précédents et utiliser plutôt la méthode avec with.

Exercice 3 : lecture d'un fichier texte en gérant les exceptions

1. Vous allez créer un fichier texte comprenant les informations suivantes (extrait de wikipedia) :

Python (prononcé /pi.tɔ̃/) est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Le langage Python est placé sous une licence libre proche de la licence BSD et fonctionne sur la plupart des plateformes informatiques, des smartphones aux ordinateurs centraux, de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS, et peut aussi être traduit en Java ou .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Il est également apprécié par certains pédagogues qui y trouvent un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation. Selon l'Index TIOBE, notamment en raison de son efficacité pour l'apprentissage automatique, sa popularité va croissante ; et en 2022 n'a toujours pas montré de signe de ralentissement.

2. Créer une fonction lecture1 qui permet de lire un fichier dont le nom est passé par argument et afficher le fichier ligne par ligne.

3. Proposer une fonction lecture2 qui permet de compter combien de fois un mot passé en argument est présent dans un fichier passé également en argument.

4. À présent vous allez faire les tests suivants en utilisant la fonction lecture1. Pour chaque test, vous allez noter les exceptions obtenues :
    a. Lire un fichier qui n'existe pas
    b. Ecrire un fichier existant mais auquel vous n'avez pas d'accès
    c. Voici quelques erreurs possibles avec les exceptions liées aux fichiers :
        i. ***FileNotFoundError*** : levée lorsque le fichier spécifié est introuvable.
        ii. ***PermissionError*** : levée lorsque l'utilisateur n'a pas les autorisations nécessaires pour effectuer l'opération de fichier demandée.
        iii. ***IOError*** : levée lorsqu'une opération de lecture ou d'écriture échoue.
        iv. ***OSError*** : levée lorsqu'une erreur système se produit, telle qu'un disque plein.

5. Gérer ces exceptions en appliquant les modifications nécessaires.

6. Pour finir vous allez faire du traitement de chaîne en utilisant les expressions régulières. Voici un exemple que vous pouvez ajouter en copiant-collant la fonction lecture1 en lecture3. Appliquer le code tel que décrit et expliquer ce que vous avez obtenu et ce que fait le code.

```
# import du package re
# Ouverture du fichier
# Pour chaque ligne du fichier:
#     Je découpe le fichier mot par mot (split)
#     Pour chaque mot:
          regex = re.compile("^([(+-)?(\d)+\.?(\d)*)$")
          if regex.match(mot) is not None:
               print("C'est un ??????")
          else:
               print("Ce n'est pas un ?????")
```

Essayer de comprendre l'expression régulière `"^([(+-)?(\d)+\.?(\d)*)$"`.

Exercice 4 : expressions régulières

1. Quelle est la différence entre **re.match()** et **re.search()**
2. Écrire la fonction `prenom_composer(prenom) -> bool` qui retourne vrai si le prénom est un prénom composé, par exemple jean-claude.
3. Écrire la fonction `is_domaine(domaine) -> bool` qui prend en argument un nom de domaine (ex : google.fr) et retourne vrai si la chaîne de caractères correspond au format d'un nom de domaine. La dernière partie du nom de domaine est composée de deux ou trois caractères.
4. À partir du fichier utilisateur.txt (présent sur moodle), rechercher combien d'utilisateurs ont pour interpréteur, bash (schéma /bin/bash)
5. À partir de ce même fichier afficher les utilisateurs qui ont comme interpréteur, l'interpréteur bash.
6. À partir de ce même fichier afficher les utilisateurs qui ont deux lettres "o" dans leur nom.
7. Écrire une fonction qui vérifie qu'un mot de passe contient au moins deux majuscules, trois chiffres et un des caractères spéciaux suivants (+- ?.*)