Exercices

Variables et opérateurs

Exercice 1

Écrivez le script pour calculer le volume d'un rectangle dont la largeur, la longueur et la hauteur soit données par l'utilisateur.

Exercice 2

Écrivez deux scripts différents qui inversent les valeurs de deux variables. Ces valeurs sont données par l'utilisateur.

Exercice 3 (en plus)

Écrivez le script qui convertit un nombre entier de secondes entré par l'utilisateur en un nombre d'années, de mois, de jours, d'heures, de minutes et de secondes. Pour une raison de simplicité, nous considérons qu'une année est constitué de 365 jours et un mois de 30 jours.

Exercice 4

Écrivez un script qui trouve toutes les occurrences d'une sous-chaîne dans une chaine en ignorant la casse

```
chaine = "Bienvenue en France. La france c'est génial, n'est-ce pas ?"
```

Exercice 5

Écrivez un script qui inverse les caractères d'une chaine de caractères entré par l'utilisateur par deux méthodes :

- le slice
- la fonction reversed()

Conditions et tests

Exercice 6

Écrivez l'algorithme puis le script Python calculent la remise suivante à un montant de type réel entré par l'utilisateur : il est accordé une remise de 5 % pour tout montant compris entre 100 et 500 € et 8 % au-delà. Pensez à tester votre script avec tous les cas possibles.

Exercice 7

Écrivez l'algorithme puis le script Python faisant saisir à l'utilisateur trois entiers, i, j et k, et les triant par ordre croissant et les afficher pour vérifier votre tri.

Exercice 8

Écrivez l'algorithme puis le script Python qui déduit le signe du produit de deux réels entrés par l'utilisateur sans utiliser la multiplication ni aucun autre calcul.

Exercice 9 (en plus)

Écrivez un algorithme qui détermine si une année entrée par l'utilisateur est bissextile. Une année bissextile est un entier divisible par quatre seulement s'il ne représente pas une année de centenaire (2000, 1900, 1800, etc.). Dans ce cas, l'entier doit être également divisible par 400. Codez le script Python correspondant.

Les boucles

Exercice 10

Écrivez un algorithme qui affiche les vingt premiers termes de la table de multiplication en signalant les multiples de 3 avec un astérisque. Codez le script Python correspondant.

Exercice 11

Écrivez un algorithme qui calcule la multiplication de deux entiers entrés par l'utilisateur sans utiliser l'opérateur * (i.e. avec des additions successives). Codez le script Python correspondant.

Exercice 12

Écrivez un algorithme qui récupère plusieurs fois une chaîne de caractères entrée par l'utilisateur et en affiche sa longueur jusqu'à ce que cette entrée corresponde au mot "end". Codez le script Python correspondant.

Exercice 13

Écrivez un algorithme qui détermine si une chaîne est un palindrome, c'est-à-dire un mot qui se lit aussi bien de gauche à droite que de droite à gauche. Codez le script Python correspondant.

Les tableaux

Exercice 14

Écrivez un algorithme qui calcule le nombre d'occurrences d'une valeur entrée par l'utilisateur dans un tableau (le nombre de fois que la valeur apparaît dans le tableau)

```
mots=["ordinateur","cafe","chocolat","digestion","cafe","formation","television"]
```

Exercice 15

Écrivez un algorithme qui calcule la moyenne des valeurs d'un tableau d'entiers.

Exercice 16

Écrivez un algorithme qui réalise l'inversion des éléments d'un tableau sans utiliser de tableau intermédiaire.

Les sous-programmes

Exercice 17

Écrivez un algorithme qui calcule dans un sous-programme la surface d'un cercle.

Exercice 18

Écrivez un algorithme qui calcule dans un sous-programme le volume d'une boîte. Codez le script Python correspondant en utilisant des valeurs par défaut pour les paramètres de la fonction.

Exercice 19

Écrivez un algorithme qui calcule et retourne dans un sous-programme la valeur la plus grande d'un tableau entre celles des deux indices passés en paramètres. Codez le script Python correspondant.

Exercice 20

Écrivez un algorithme avec un sous-programme qui retourne le nom du mois en fonction de son numéro. Par exemple, si l'utilisateur entre 1, il sera affiché janvier. Codez le script Python correspondant.

Exercice 21

Écrivez un algorithme qui calcule le nombre de mots d'une chaîne de caractères avec un sous-programme. Nous considérons que deux mots sont uniquement séparés par un espace pour des raisons de simplicités. Codez le script Python correspondant.

Exercice 22 (en plus)

Écrivez l'algorithme entier puis le code Python pour le jeu du morpion en le découpant en sous-programme :

- **initGrille(grille)** : grille.
- **grilleComplete(grille)** : boolean.
- **winner(grille)** : boolean.
- **afficheGrille(grille)**.
- **saisirJoueur(grille) : int,int** (un joueur ne doit pas pouvoir tricher et il doit rentrer un int entre 0 et 2 compris).

```
| | | |
| | | |
| | | |
Entrer un entier entre 0 et 2 pour x
0
Entrer un entier entre 0 et 2 pour y
1
| |X| |
| | | |
| | | |
```

Exercice 23 (en plus)

Codez en Python le tri à bulles. Utiliser cet exemple :

```
tab = [98, 22, 15, 32, 2, 74, 63, 70]
```

Les fichiers

Exercice 24

Ecrivez l'algorithme qui parcours arborescence à partir d'un répertoire racine et affiche les sous-dossiers et fichiers récursivement

En plus : vous n'afficherez que le nom des dossiers et fichiers avec une indentation et une étoile (*) quand c'est un fichier

Exercice 25

Écrivez l'algorithme qui génère automatiquement dans un fichier texte les tables de multiplication bien écrites de 1 à 20.

Exercice 26

Écrivez l'algorithme qui recopie un fichier texte dont le nom est donné par l'utilisateur dans un autre ficher texte nommé "copie.txt".

Exercice 27

Écrivez l'algorithme qui recherche et affiche la ligne la plus longue dans un fichier texte.

Exercice 28

Codez le script qui lire et interpréter un fichier au format CSV (sans utiliser CSVReader) pour afficher chaque élément séparément

```
ID;NOM;PRENOM;MANAGER
15;BESSA;Hydris;Y
17;FREBAULT;Pierre;N
18;GINOUX;Marine;N
21;TERBAH;Dorian;N
25;SULTAN;Eric;N
```

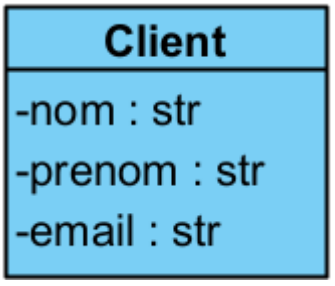
Affichage dans la console :

```
ID=15 - NOM=BESSA - PRENOM = Hydris - MANAGER = True
```

Les classes

Exercice 29

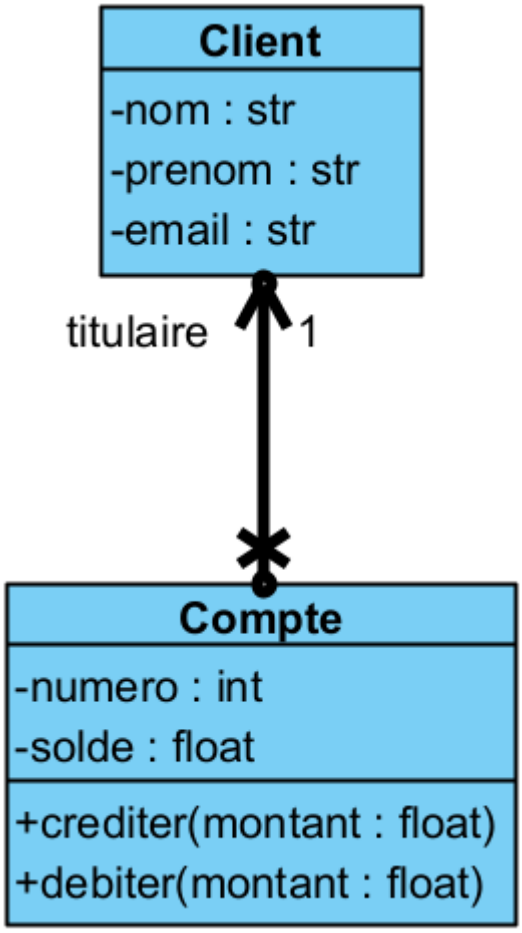
Créer une classe Client dans le module mclient qui est inclut dans le package pclient



- tous les attributs sont privés
- l'email est fourni à la construction de l'instance et ne peut être modifié
- le nom et le prénom pourra être indépendamment renseignés soit à la construction de l'objet soit via des setters
- créer un module mainBanque qui instanciera deux clients avec nom, prénom et email
- y afficher le contenu (print)

Exercice 30

Créer une classe Compte dans le module mcompte qui est inclut dans le package pcompte



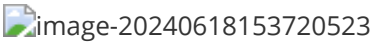
- tous les attributs sont privés
- le numéro et le solde sont fournis à la construction de l'instance et ne peuvent être modifiés
- le titulaire pourra être modifié à postériori
- modifier le module mainBanque en y ajoutant la création de comptes reliés au titulaire
- y afficher le contenu (print)

Exercice 31

- Dans la classe Compte, rajouter une variable de classe privée nbr_comptes initialisée à 0
- A chaque création de compte, incrémenter ce compteur
- Rajouter également une méthode de classe get_nbr_comptes() qui renvoie la valeur du compteur
- Rajouter un destructeur à la classe Compte qui va décrémenter le compteurClient

L'héritage

Exercice 32



Adapter la classe Compte :

- pour y rajouter une date_ouverture de type date qui est passée au constructeur ou mise à la date du jour par défaut
- modifier la méthode débiter pour qu'elle envoie un bool si le débit n'a pu être effectué (solde insuffisant)

Créer une classe CompteEpargne qui hérite de Compte avec un taux_interet et duree_blocage en plus et coder les méthodes :

- calcul_interet() : ajout des intérêts annuels au solde du compte
- debiter() : verifier que la durée de blocage est dépassé et vérifier que le montant à débiter est autorisé (pas de découvert)

Gestion des exceptions

Exercice 33

- Créer dans un package pbanque, un module mexception qui contiendra les exceptions applicatives
- Créer une classe d'exception BanqueException (hérite d'Exception)
- Créer une spécialisation BanqueSoldeException et BanqueBlocageException qui hérite de BanqueException
- Modifier les méthodes débiter pour retirer le booléen et déclencher à la place des exceptions pour le solde insuffisant et le blocage du compte
- Dans mainBanque, tester les cas d'erreurs en attrapant les Exception spécifiques et renvoyer un message à l'utilisateur

Les collections

Exercice 34

- Créer dans package **pdao**, in, module **mdao** qui contiendra une classe **BaseDao** :

```
class BaseDao :
    def __init__(self):
        pass

    def findAll(self):
        raise NotImplementedError()

    def findById(self, id):
        raise NotImplementedError()

    def create(self, obj):
        raise NotImplementedError()

    def update(self, obj):
        raise NotImplementedError()

    def delete(self, obj):
        raise NotImplementedError()
```

- Créer un nouveau module **mdaomemory** qui contiendra les Dao Client et Compte :

```
class ClientDaoMemory(BaseDao):

    def __init__(self):
        super().__init__()
```

```
        self.__clients = []

    def findAll(self): # renvoie le tableau de clients
        pass

    def findById(self, id): # recherche le client par son email
        pass

    def create(self, obj): # rajoute le client (obj) dans le tableau si email
n'existe pas déjà (-> Exception)
        pass

    def update(self, obj): # met à jour le client (obj) dans le tableau
        pass

    def deleteById(self, id): # supprime le client en recherchant par son email
        pass

    def findAllByNom(self, nom): # renvoie le tableau de clients filtrer par nom
        pass
```

- Adapter **mainBanque** pour appeler et tester les méthodes du **ClientDaoMemory**

```
clientDao = ClientDaoMemory()
liste = clientDao.findAll()
```

Exercice 35 (en plus)

- Créer une classe **CompteDaoMemory** pour pouvoir manipuler les classes **Compte** et **CompteEpargne**
- Adapter **mainBanque** pour appeler et tester les méthodes du **CompteDaoMemory**

La persistence

Exercice 36

- Sur le même format que dans les exercices 34 et 35, coder dans un module **mdaocsv** les classes **ClientDaoCsv** et **CompteDaoCsv**
- Vous devrez utiliser *csv.reader* ou *csv.DictReader* en créant de méthode interne :
  - readAll() : qui renverra le tableau de Client (ou Compte) à partir de la lecture du fichier en csv
  - writeAll(liste) : qui écrira dans le fichier en csv la liste des objets Client (ou Compte)
- Les méthodes *findAll(), findById(id)*, etc ... réutiliseront toutes la méthode readAll() et writeAll()
- Tester le bon fonctionnement de ces nouveaux Dao dans **mainbanque**

Exercice 37

- Créer un module msqlite qui contiendra un **main**:
- Se connecter à une base **banque.db** via sqlite3 et exécuter cette requête de création :

```
CREATE TABLE IF NOT EXISTS client(
    email TEXT PRIMARY KEY,
    nom TEXT,
    prenom TEXT)
```

- Tester l'extension database pour voir la table créé dans ls BDD sqlite
- Fermer la connection précédente
- Réouvrire une nouvelle connection pour créer puis mettre à jour un client :

```
INSERT INTO client (email, nom, prenom) VALUES (?,?,?)
```

```
UPDATE client SET nom = ?, prenom = ? WHERE email = ?
```

- Valider la connection (commit) et la fermer
- Réouvrire une nouvelle connection pour créer pour lister les clients :

```
SELECT email, nom, prenom FROM client
```

Exercice 38

- Créer dans le module mdaosql la classe ClientDaoSql et tester

```
class ClientDaoSql(BaseDao):
    def __init__(self, bdd: str):
        self._bdd = bdd

    def findAll(self):
        with sqlite3.connect(self._bdd) as conn:
            conn.execute("""
                SELECT email, nom, prenom FROM client
            """)
            ...
```

Les tests

Exercice 39

- Créer un test unitaire qui va tester le bon fonctionnement de la mise à jour d'un client en utilisant ClientDaoSql
  - Vous utiliser soit unittest soit pytest
  - Si pytest: penser à installer dans votre venv via la commande : pip install pytest
  - Remarque : pênsez aux AAA :
    - Arrange : Créer un client
    - Act : mise à jour du client
    - Assert : vérification des données mis à jour

SQLAchemy

Exercice 40

1. Créer un nouveau venv banque
2. ouvrir vs code et sélectionner ce nouveau venv
3. installer sqlachemy dans le venv :

```
pip install sqlachemy
```

4. créer dans un package et un module qui contiendra les classes Client et Compte
5. Faire le mapping qui permettra d'arriver à ce script SQL

```
CREATE TABLE IF NOT EXISTS customer(
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            email TEXT UNIQUE NOT NULL,
                            last_name TEXT,
                            first_name TEXT);

CREATE TABLE IF NOT EXISTS account(
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            reference INTEGER UNIQUE NOT NULL,
                            type TEXT NOT NULL,
                            amount FLOAT,
                            start_date DATE,
                            owner_id INTEGER,
                            FOREIGN KEY(owner_id) REFERENCES customer (id));
```

```
if __name__ == "__main__":
    engine = create_engine("sqlite:///test.db", echo=True)
    Base.metadata.create_all(engine)
```

Exercice 41

- Créer un ClientDaoSA qui prendra en paramètre l'engine SQLAlchemy
    - Développer les méthodes
        - findAll : avec session.scalars(stmt)
        - findById : avec session.get()
        - create : avec session.add()
        - update : avec le principe du dirty checking ou via session.merge()
        - delete : avec le session.delete()
- Tester le bon fonctionnement