Modularité et mise au point des programmes

Petite anecdote : Toyota a rappelé 600k voitures suite à un bug informatique.

L'écriture du code ne représente pas la part la plus importante du dev. Le debug et la maintenance sont beaucoup plus important qu'on ne pourrait s'en douter.

- Module et documentation

Que l'on distribue son code ou que l'on utilise celui déjà existant, la manipulation des modules et de la documentation font partit intégrante du dev logiciel.

1 – Programmation modulaire

- Principes généraux

Le principe de modularité est particulièrement important dans le dev :

- Simplifie les tests
    - Plus simple d'écrire des jeux de test fonction par fonction, si elles sont bien découpées
- Permet de réutiliser du code
    - Par exemple, math ne regroupe que des fonctions liées aux math
- Facilite la maintenance
    - Si bug dans le fichier, plus facile à trouver

Pour ceux faire, un dev peut :

- Découper le code en fonctions
- Grouper les fonctions dans une classe, que l'on appelle alors ***méthodes***
- Grouper les fonctions par thèmes, dans un fichier séparé
- Grouper des fichiers en bibliothèques

La prog modulaire intervient :

- Lors de l'écriture de portions de code afin de le réutiliser plus tard, ou le distribuer
- Lors de l'utilisation, pour répondre à un besoin, de code écrit par un autre

- Modularité en Python

On peut, et on doit, écrire des fonctions et des procédures, et les réutiliser

Python nous permet de faire de la prog orienté objet et donc, de grouper des fonctions (***méthodes***) au sein d'une classe, selon le type d'objet sur lequel elles agissent

Exemple d'un jeu, on va regrouper tout ce qui touche au personnage du joueur dans une classe, ce qui touche aux mobs dans une autre, aux objets dans une autre, etc...

Fonctions, procédures et classes peuvent être groupées par thèmes dans des fichiers séparés alors appelé ***modules*** qui eux même peuvent être groupés en ***packages***

Exo :

Créer un nouveau projet sous PyCharm, appelé modularite

On supprime tout ce qu'il y'a dans la fenêtre

Clic droit sur dossier Modularite / ajouter Python file / appelé operations

Dans le fichiers operations.py :

```
def multiplication(nb, multiple) :
      print(nb*multiple)

def division(nb, diviseur) :
      print(nb/diviseur)
```

2- Importation des modules

Certains modules Python sont installés par défaut (bibliothèque standard) et d'autres peuvent être ajoutés en utilisant des outils comme par exemple *pip*.

On peut trouver la plupart des modules tiers sur le dépôt Pypi (Python Package Index, https://pypi.org/)

Pour importer un module, on utilise le mot-clé ***import***.

Ex :

Il est possible d'importer un module préinstallé, comme le module ***random*** en écrivant :

Import random

Et ensuite l'utiliser :

A = random.randint(1, 10)

On peut aussi importer un module et le renommer, en général avec un nom plus court :

Import random as rnd

A = rnd.randint(1, 10)

Ou encore, n'importer qu'une fonction d'un module :

From random import randint

A = randint(1, 10)

Importer tout le contenu d'un module en utilisant * peut paraitre efficace pour un dev débutant.

En réalité, on ne maitrise plus exactement la liste des imports, ce qui peut êtr esource de bugs.

Si l'on vient à reprendre plus tard son travail ou à le distribuer, il devient plus compliqué de garder une trace simple de l'endroit où se trouve les fonctions et classes utilisé, rendant la maintenance plus difficile.

Utiliser la forme import * est à bannir.

On va importer le module créé précedemment :

Comment appelé les 2 fonctions dans le fichier main.py ?

```
import operations as ope

ope.multiplication(5, 3)
ope.division(5, 3)
```

3 – Documentation

Que l'on écrive ou utilise un module, sa documentation est indispensable pour que le travail soit réutilisable.

Parmi les différents mécanismes, l'un des plus simple est le ***docstring***, qui peut être rattaché à une fonction, une procédure, une méthode, une classe, un module ou un package.

Dans tous les cas, elle doit figurer au début de l'entité qu'elle commente.

Cette documentation devient ensuite disponible via la commande ***help***

Ex :

Help(random)

Help(print)

Exo :

```
def factorielle(n):
            """
      Calcul de la factorielle :
     n! = 1x2x...xn
      #    >>> factorielle(5)
         120
        """
        res = 1
            for i in range(2, n+1):
                  res = res * i
             return res

print(factorielle(5))
help(factorielle)
```

Faire la doc des 2 fonctions :

```
      def multiplication(nb, multiple) :
      """
      Effectue la multiplication d'un nombre par un autre
      """
            print(nb*multiple)

      def division(nb, diviseur) :
            """
      Effectue la division d'un nombre par un autre
      """
            print(nb/diviseur)
```

Quiz :

Parmi les lignes suivantes, laquelle ne permet pas d'importer et d'utiliser la totalité du module itertools :

A – import itertools

B – import itertools as itt

C – from itertools import cycle

On souhaite écrire une portion de code qui permette de savoir si une année est bissextile, quel est le fonctionnement le plus approprié :

A – faire un prog principal qui demande à l'user de taper une année et qui indique si elle est bissextile

B – faire une fonction qui indique si une année est bissextile, en la passant en paramètre, et renvoie un booléen

C - écrire un module bissextile.py qui contiendra tout ce qu'il faut pour tester si l'année est bissextile

Un fichier arithmetique.py contient des fonctions sur l'arithmétique des nombres entiers, en particulier :

- Expo qui prend en paramètres les entiers a, b, n et calcule le reste de la division par n de a puissance b
- Pgcd qui prend en paramètres les entiers a et b et renvoie le plus grand diviseur commun de a et b
- Decomposition qui prend un entier positif n en paramètre et renvoie la liste de ses facteurs premier ainsi que que leur multiplicité

A – Proposer des docstrings pour le module et les trois fonctions décrites (il n'est pas demandé de donné le code des fonctions)

B – ajouter quelques tests aux docstrings de la question a