Exercice de révision

# Objectif(s)

Révision générale des concepts de base de la programmation procédurale.

# Exercice 1 : Débogage - 30 min

Dans le fichier **debug\_deviner\_prix.py**, vous trouverez un programme qui permet de manipuler des fruits et leurs prix et de deviner le prix d’un fruit choisi aléatoirement parmi la liste des fruits. Malheureusement, cette application est défectueuse et c’est à vous de la réparer. Voici les informations à fournir pour chaque erreur trouvée.

**Erreur**

Mauvais code :

# Dictionnaire des produits (fruits) avec leurs prix  
fruits = [  
 "pomme": 1.50,  
 "banane": 0.75,  
 "orange": 1.20,  
 "fraise": 2.00,  
 "kiwi": 1.80  
 ]

Bon code :

# Dictionnaire des produits (fruits) avec leurs prix  
fruits = {  
 "pomme": 1.50,  
 "banane": 0.75,  
 "orange": 1.20,  
 "fraise": 2.00,  
 "kiwi": 1.80  
}

Explications : Un dictionnaire n’utilise pas des crochets.

**Erreur**

Mauvais code :

prix = float(input(f"Entrez le prix de {fruit.capitalize()} : ")

Bon code :

prix = float(input(f"Entrez le prix de {fruit.capitalize()} : "))

Explications : Il manquait une parenthèse pour fermer la chaine.

**Erreur**

Mauvais code :

if devine = prix\_correct:

Bon code :

if devine == prix\_correct:

Explications : Pour comparer deux variables, il faut mettre deux =, sinon ça fait que dire que les deux variables sont identiques.

**Erreur**

Mauvais code :

if choix == '1':  
 deviner\_prix()  
 if resultat == True :

Bon code :

if choix == '1':  
 resultat = deviner\_prix()  
 if resultat == True :

Explications : La variable résultat n’existait pas et n’avait pas de valeur avec de l’avoir mis dans la comparaison, donc l’if n’avait pas de valeur pour comparer le True.

**Erreur**

Mauvais code :

essais += 1

Bon code :

essais -= 1

Explications : En mettant un plus, le nombre d’essais restants augmente à chaque tour, ce qui fera en sorte que le jeu ne terminer pas après 3 essais comme prévu.

**Erreur**

Mauvais code :

fruit = input("Entrez le nom du fruit à ajouter : ").capitalize()  
if fruit not in fruits:  
 print("Ce fruit existe déjà. Utilisez l'option pour modifier le prix.")

Bon code :

fruit = input("Entrez le nom du fruit à ajouter : ").lower()  
if fruit in fruits:  
 print("Ce fruit existe déjà. Utilisez l'option pour modifier le prix.")

Explications : En mettant not in, on fait en sorte que même si le fruit n’est pas dans la liste, le message qui affiche est quand même celui que le fruit existe déjà. De plus, le capitalize ne permet pas de bien vérifier les fruits dans la liste, puisqu’ils sont en minuscules. Donc, mettre lower permet de vérifier correctement les fruits.

**Erreur**

Mauvais code :

fruit\_modif = int(input("Entrez le nom du fruit à modifier : ").lower())

Bon code :

fruit\_modif = input("Entrez le nom du fruit à modifier : ").lower()

Explications : Le int transforme le string entré en chiffre, même si on demande un mot, donc il est nécessaire de l’enlever.

**Erreur**

Mauvais code :

if prix >= 0:  
 print("Le prix doit être positif.")

return

Bon code :

if prix < 0:  
 print("Le prix doit être positif.")

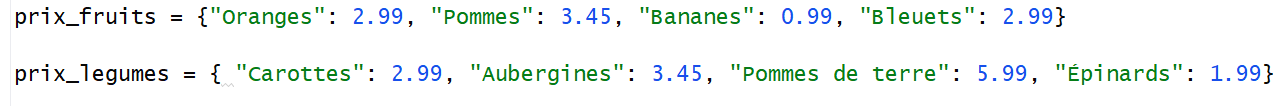
return

Explications : La manière originale faisait en sorte que le prix était considéré comme étant négatif s’il était plus grand que 0, ce qui est illogique. Il faut donc échanger le symbole pour faire en sorte qu’on vérifie correctement le prix entré. De plus, le return doit être indenté, car sinon, il retourne toujours le message d’erreur même s’il n’y a pas d’erreur.

# Exercice 2 : Pseudo code ou organigramme - 30 min

Pour faire tes courses de la semaine, tu disposes d’une liste d’achat qui contient des fruits et/ou des légumes et d’une somme d’argent. Tu veux acheter une livre de chaque fruit/légume de la liste. Tu disposes aussi du prix des fruits et des légumes par livre. Écris un pseudo code ou un organigramme pour trouver tous les fruits / légumes que la somme d’argent dont tu disposes te permet d’acheter.

Exemple :



**Les entrées fournies au programme** :





**Sortie du programme :**



DEBUT

dict\_fruit <- {}

dict\_legume <- {}

AFFICHER "Veuillez entrer le produit à acheter"

LIRE produit

TANT QUE produit <> ""

AFFICHER "Veuillez entrer le produit à m-acheter"

LIRE produit

SI produit dans dict\_fruit ou dict\_legume

list\_produits <- produit

SINON

AFFICHER "Le produit n'existe pas dans les dictionnaires"

FIN SI

AFFICHER "Veuillez entrer le produit à acheter"

FIN TANT QUE

AFFICHER "Veuillez entrer la somme d'argent"

LIRE somme

SI somme < 0 :

AFFICHER « Cette somme est invalide »

Redemander la somme

FIN SI

list\_produit\_effectif <-[]

i <- 0

total <- 0

SI list\_produits != vide

TANT QUE total < somme

SI list\_produits[i] dans dict\_fruits

prix = dict\_fruits[list\_produits[i]]

SINON

prix = dict\_legumes[list\_produits[i]]

FIN SI

total <- total + prix

SI total < somme

lsit\_produit\_effectif <- list\_produits[i]

FIN SI

i <- i + 1

FIN TANT QUE

FIN SI

AFFICHer list\_achat\_effective

FIN

# Exercice 4 : Code Python -75 min

En se basant sur l’énoncé et en analysant le logigramme ci-dessous, programmez l’application qui est représentée.

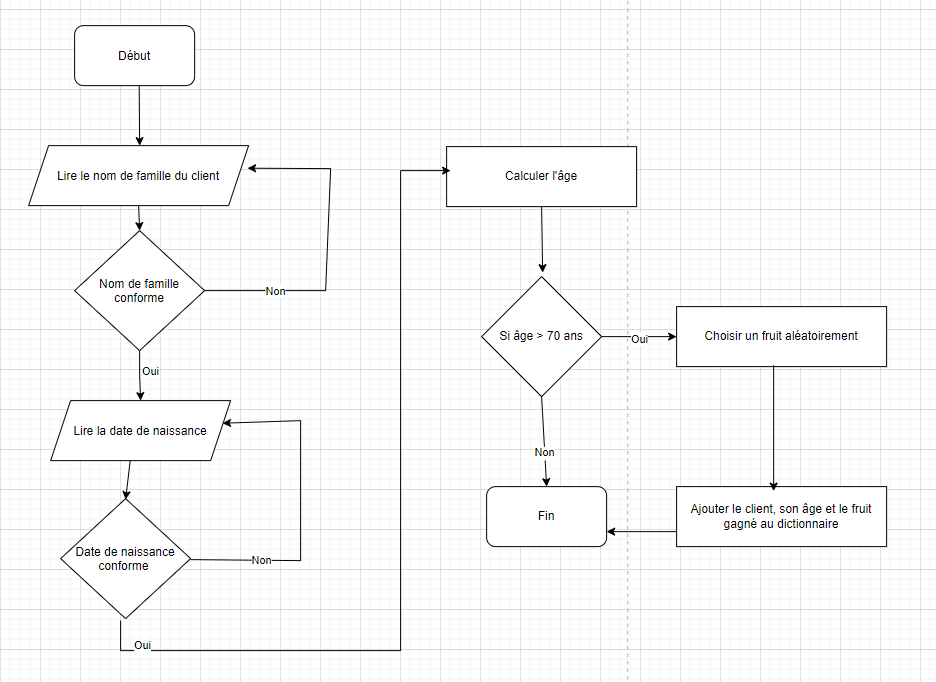
**Offrir un fruit au client :**

Vous devez créer une application qui permet d’offrir un fruit choisi aléatoirement à tout client âgé de plus de 70 ans.

* Le fruit est choisi aléatoirement parmi les fruits du dictionnaire :

Fruits = {  
 "pomme": 1.50,  
 "banane": 0.75,  
 "orange": 1.20,  
 "fraise": 2.00,  
 "kiwi": 1.80  
 }

* Le nom de famille du client doit respecter les critères suivants :
  + Commencer par une lettre majuscule
  + Avoir au moins 3 lettres de longueur
  + Contenir seulement des lettres, des espaces " ", des tirets "-" et des apostrophes "'"
* Sa date de naissance doit être une date valide, au format "JJ-MM-AAAA".
* Utilisez *Datetime* pour calculer son âge en années.
* Toutes les caractéristiques du client gagnant doivent être enregistrées dans un dictionnaire.



# Exercice 4 : Pytest - 15 min :

Dans le fichier **trouver\_prix.py**, vous trouverez une fonction. Lisez sa documentation, comprenez le code et son exécution. Ensuite, dans le fichier **test\_trouver\_prix.py**, écrivez une fonction de test appropriée. Utilisez le module *pytest* pour faire des jeux d’essais pertinents et variés (au moins 5). Il est attendu que vous fassiez des jeux d’essais qui couvrent toutes les conditions.