

Fiche de TD – Python

Les supports de cours, ainsi que plusieurs exemples de code, sont disponibles sur notre EPI (<https://cours.univ-paris1.fr/fixe/06-M1-SI-et-Informatique>) et sur le Repl.it dédié au cours (<https://replit.com/@ManueleKirsch/SystemeInformationInformatique>).

Pour la réalisation de cette fiche, pensez à regarder d'abord les vidéos d'introduction à Python et d'introduction à l'environnement Repl.it qui sont aussi disponibles sur l'EPI. Vous pouvez réaliser la fiche aussi bien sur l'environnement Repl.it que sur un des IDEs indiqués dans la vidéo d'introduction (Idle, Visual Studio Code, ou PyCharm).

TD A :

Exercice pas à pas :

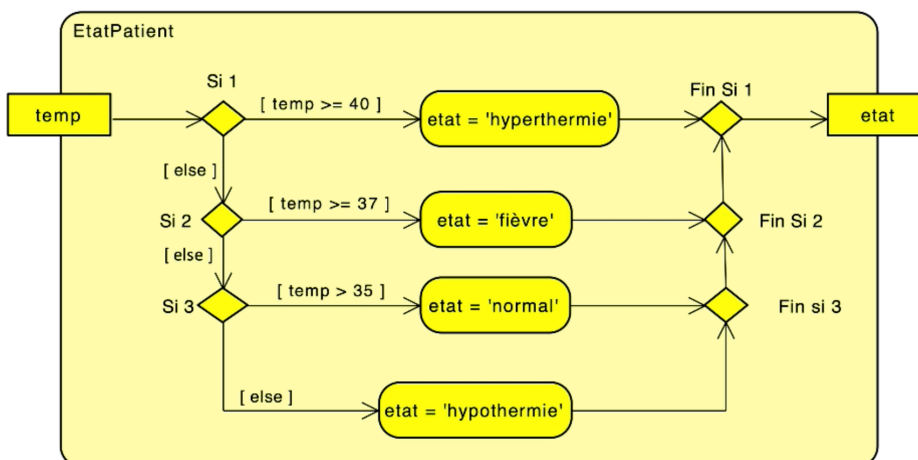
Pour ce premier exercice, nous allons reprendre un algorithme que nous avons travaillé en cours, celui permettant d'établir l'état d'un patient en fonction de sa température, et nous allons le mettre en œuvre à l'aide du langage Python.

Pour savoir dans quel état se trouve un patient, nous devons considérer sa température. Si la température est supérieure (ou égale) à 37, on a de la fièvre. Au-delà de 40, l'état est d'hyperthermie. Si la température est inférieure (ou égale) à 35, le patient est en hypothermie. En dehors de ces cas, le patient est dans un état normal.

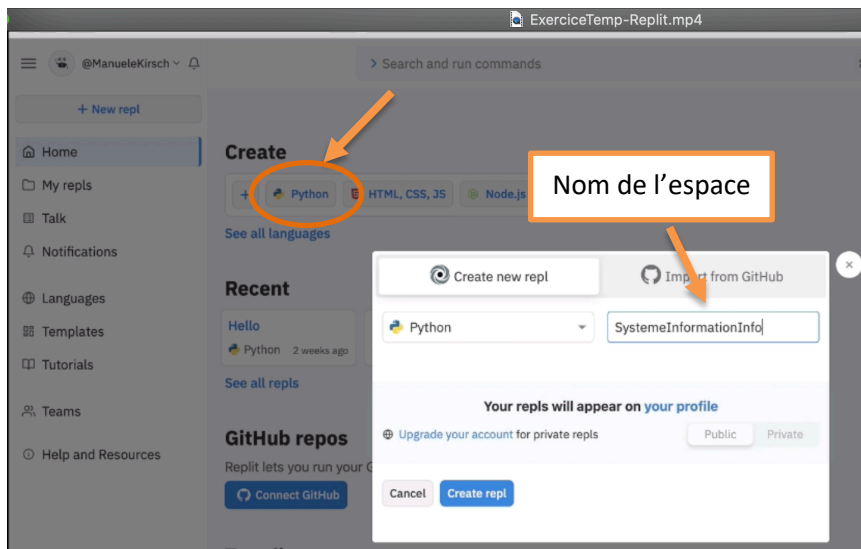
Ce premier exercice représente un tutoriel, qui vous pouvez faire seul (en regardant la vidéo qui est disponible sur l'EPI) ou accompagné de votre chargé de TD. Lisez avec attention les instructions et suivez-les pas à pas.

Nous allons commencer donc par rappeler l'algorithme qu'on avait fait en cours. À cette occasion, on avait défini une entrée (la température) et une sortie (l'état du patient) :

On y voit une séquence de « **si – sinon – si – sinon** » qui nous permettent de tester les différents paliers de température (≥ 40 , entre 37 et 40, entre 35 et 37 et ≤ 35).



Il nous reste qu'à trouver une valeur pour la variable **temp** (nous pouvons demander à l'utilisateur de la renseigner) et d'imprimer la valeur trouvée pour la variable **etat**.



Une fois, l'algorithme revu, nous pouvons passer au code.

Pour cela, nous allons utiliser la plateforme Repl.it. Allez donc sur <http://replit.com> et créer votre compte. Ensuite, il faut créer votre espace « replit », c.a.d. votre espace de travail où vous allez faire vos exercices.

Une fois votre espace créé, nous allons créer un nouveau fichier (temperature.py) pour notre exercice.

Nous allons réaliser notre code sur ce fichier qu'on vient de créer.

La première étape pour notre code est donc de demander à l'utilisateur une valeur qu'on gardera dans la variable « **temp** ». Pour cela, nous allons utiliser la fonction « **input** », laquelle permet de lire un texte en entrée à partir du clavier.

```
temp = input('Temperature ? ')
```

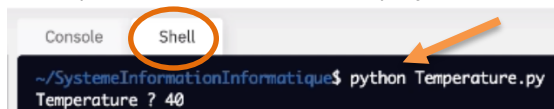
Problème : la fonction **input** lit une **chaîne de caractères** et pas une valeur numérique. Or ce qu'il nous faut, c'est justement une valeur numérique (un numéro réel) pour représenter la température de notre patient. Nous allons donc devoir **convertir** la chaîne qu'on reçoit de la fonction « **input** » à l'aide de la fonction « **float** », laquelle permet de convertir une chaîne de caractères (un string) en float.

```
temp = float(input('Temperature ? '))
```

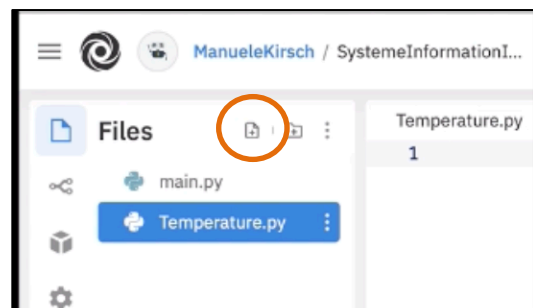
Afin d'être sûr que notre entrée marche bien, nous allons afficher le contenu de la variable « **temp** » à l'aide de la fonction « **print** ».

Pour tester notre code, nous allons nous servir de l'onglet « **console** » qui se trouve à droite sur Replit.

En cliquant sur la « console » (qui fonctionne comme un terminal), nous allons pouvoir taper « **python Temperature.py** » (« **Temperature.py** » étant le nom de notre fichier). **Attention** : majuscule et minuscules sont différentes (T ≠ t).



Maintenant que nous avons tester l'entrée des données et que nous sommes sûrs que la variable « **temp** » contient une valeur, nous allons pouvoir passer à l'algorithme que nous avons fait en cours. Afin de traduire la structure de « **si – sinon si – sinon** » que nous avons proposé dans l'algorithme, nous allons nous servir de la structure « **if – elif – else** » disponible sur Python. Elle va nous permettre de traduire directement la séquence de « **si** » proposée dans l'algorithme. A chaque « **if** » et « **elif** », on associe une condition, qui va donc vérifier un de nos paliers.



```
Temperature.py
1 # Exercice sur la temperature d'un patient
2
3 temp = float(input('Temperature ? '))
4
5 print(temp)
```

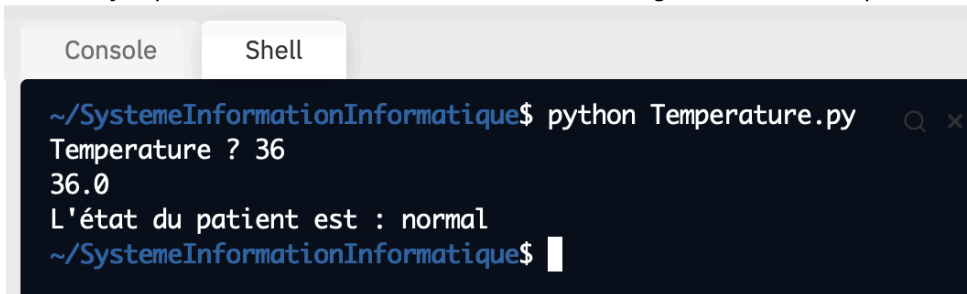
```
7 if temp >= 40 :
8     | etat = 'hyperthermie'
9 elif temp >= 37 :
10    | etat = 'fièvre'
11 elif temp > 35 :
12    | etat = 'normal'
13 else :
14    | etat = 'hypothermie'
```

Ainsi, à chaque niveau, nous testons un état possible : d'abord l'hyperthermie (si ça dépasse 40°C, avec la ligne « **if temp >= 40 :** »), ensuite, si ce n'est pas le cas (« **sinon** » donc), on teste dans le premier « **elif** » l'état de fièvre (et donc si ça dépasse 37°C, avec la ligne « **elif temp >= 37 :** »). Nous n'avons pas besoin de tester dans celui-ci si température est < 40°C, puisque nous sommes dans le « **sinon** » du premier test. Nous continuons ensuite sur le prochain palier, à l'aide d'un deuxième « **elif** », laquelle va nous permettre de distinguer entre un état normal (pour une température > 35 °C) et l'hypothermie. Ce dernier entre ainsi dans notre « **else** », correspondant au cas où aucune des conditions précédentes ne serait vraie. **Faites attention** à ne pas oublier le « **:** » dans les « **if** », « **elif** » et « **else** » (pour démarrer chaque bloc) et de bien respecter le bon nombre d'espaces blancs (**ne pas mélanger les tabulations et les espaces blancs**).

A la fin du bloc « **if – elif – else** », nous avons avoir l'état de notre patient enregistré dans la variable « **état** ». Il nous reste donc l'afficher pour que l'utilisateur soit au courant. La fonction « **print** » nous permettra à nouveau d'afficher une valeur à l'écran. Nous pouvons afficher plusieurs valeurs sur une même ligne à l'aide de la fonction « **print** ». Pour cela, il suffit de séparer les valeurs avec une virgule :

```
print ("L'état du patient est :", etat)
```

Il nous enfin qu'à tester notre code en l'exécutant sur l'onglet « shell » de Replit.



```
~/SystemeInformationInformatique$ python Temperature.py
Temperature ? 36
36.0
L'état du patient est : normal
~/SystemeInformationInformatique$
```

Code complet :

```
# on démarre par récupérer la température du patient
temp = float(input('Temperature ? '))

# différentes formes d'organiser la séquence de tests est possible
# penser à tester toutes les possibilités
if temp >= 40 :
    etat = 'hyperthermie'
elif temp >= 37 :
    etat = 'fièvre'
elif temp > 35 :
    etat = 'normal'
else :
    etat = 'hyperthermie'

# on termine par afficher l'état du patient
print ("L'état du patient est :", etat)
```

Exercices :

- 1) Réaliser un programme mettant en œuvre un des algorithmes que nous avons travaillé lors des fiches précédentes : celui permettant aux clients de simuler les gains qu'ils pourraient avoir avec un nouvel investissement.

Une banque veut offrir à ces clients la possibilité de simuler un nouvel investissement de type épargne, le Livret Alpha+. Celui-ci offre pendant les 3 premières années un taux de 2% à l'an et de 1,5% pour les années suivantes. Le programme devra, à partir d'un montant initial renseigné par le client, afficher une prévision des gains pour les 8 prochaines années.

Suggestion : pour lire une valeur à partir du clavier, on peut utiliser la fonction « input », mais n'oublier que cette valeur sera une chaîne de caractères. Il faut donc la convertir en numéro réel :

```
montant = float (input("Montant initial ? "))
```

- 2) Faire un programme en Python qui calcule le montant d'impôt à payer par un habitant la République des Bananes. Le programme devra demander à l'utilisateur la valeur de ses revenus (salaires, primes, heures complémentaires, mais également les aides à la formation) et les garder dans un tableau. Ce tableau sera utilisé par l'algorithme que nous avons préparé lors des fiches précédentes et que nous allons maintenant implémenter en Python. A la fin du programme, celui-ci devra afficher le montant total d'impôts à payer.

Dans la République des Bananes (petit pays perdu au sein du continent américain), nous avons trois tranches d'impôts : ceux qui gagnent moins de RB\$ 9 000,00 ne sont pas imposables ; ceux qui gagnent entre RB\$ 9 000,00 et RB\$ 15 000,00 paient 15% d'impôt sur leurs revenus ; ceux qui gagnent plus de RB\$ 15 000,00 doivent payer 20% d'impôts sur leurs revenus. Par ailleurs, trois types de revenus sont imposables : les salaires, les primes et les heures complémentaires, mais uniquement si le citoyen a reçu des aides à la formation (pour les citoyens n'ayant pas reçu ces aides, les heures complémentaires ne sont pas imposables). Notre programme devra comptabiliser l'ensemble de revenus reçus par un citoyen, avant de vérifier dans quelle catégorie d'imposition il rentrerait et calculer, en fonction de cette catégorie, le montant d'impôts à payer.

Les revenus seront stockés dans un tableau contenant 4 positions : le montant de salaire reçu dans l'année ; le montant des primes reçues ; le montant des heures complémentaires reçues ; et enfin le montant d'aides à la formation reçues. Par exemple, Toto (voir figure ci-dessous) a reçu cette année RB\$ 15 000,00 de salaire, il n'a pas reçu de prime (RB\$ 0,00), mais il a reçu RB\$ 5 000,00 en heures complémentaires et a touché RB\$ 3 000,00 en aide à la formation. Ses revenus imposables s'élèvent donc à RB\$ 20 000,00 (attention, les aides à la formation, elles, ne sont pas imposables). Il doit donc payer un total de RB \$ 4 000,00 en impôt à la République des Bananes

Données renseignées :

0	1	2	3
15 000	0	5 000	3 000

→

Sortie affichée :

4 000

Suggestion : en Python, pour créer un tableau (liste) dont on connaît à l'avance la taille, il suffit de lui assigner des valeurs. Par exemple, pour créer un tableau « revenus » avec 4 positions :

```
revenus = [ 0 , 0 , 0 , 0 ]
```

- 3) Faire un programme en langage Python suivant les étapes indiquées ci-dessous :
- Le programme doit d'abord permettre à un utilisateur de rentrer plusieurs valeurs. Le programme doit indiquer si la valeur renseignée ne contient que des lettres ou s'il s'agit d'un numéro entier positif. Le programme doit s'arrêter si l'utilisateur entre la valeur « fin ».

Suggestion : utiliser pour cela les fonctions « isalpha » et « isdigit ». Par exemple, pour savoir si une variable « valeur » ne contient que des chiffres :

```
valeur.isdigit()
```

- Modifier le programme pour enregistrer dans un tableau (liste) les valeurs qui correspondraient à des numéros entiers. Afficher le tableau avant de terminer le programme (après que l'utilisateur aurait renseigné le mot « fin »).

Suggestion : utiliser méthode « append » pour ajouter une nouvelle valeur à la liste.

```
tableau.append(valeur)
```

Attention : dans cet exercice, on veut la valeur en tant que numéro entier, il faut donc la convertir à l'aide de la fonction « int(valeur) ».

- Chercher sur internet des fonctions permettant de calculer rapidement la somme, le min et le max du tableau renseigné. Afficher ces valeurs avant de terminer le programme.

- 4) Concevoir un programme en Python capable de demander à son utilisateur un numéro, puis de lui indiquer à l'écran si un numéro est pair ou impair. Utilisez pour cela l'opérateur « % » (module), lequel calcule le reste d'une division entière.

- 5) Améliorer le programme précédent afin de lui permettre de compter combien de numéros impairs existent dans un tableau. Procéder par étapes :

- Dans un premier temps, modifier le programme pour qu'il puisse demander à l'utilisateur plusieurs numéros. L'utilisateur pourra entrer autant de numéros qu'il le souhaite. Le programme arrêtera de lui en demander dès que l'utilisateur aura renseigné un n° négatif (*on se limitera pour l'exercice aux numéros > 0*). Au fur et à mesure que l'utilisateur renseigne ces numéros, on les stockera dans un tableau.

- On analyse un par un les numéros qui sont dans le tableau à l'aide du code que nous avons fait dans l'exercice précédent. On affiche le nombre de numéros impairs qu'on a pu compter dans le tableau.

Suggestion : le boucle « for » permet de parcourir la totalité des éléments d'un tableau.

```
for numero in tableau :
```

- 6) On va encore améliorer le code qu'on a produit dans la question 5), à l'aide de la notion de fonctions :

- Créer une fonction contenant uniquement l'algorithme permettant de savoir si un numéro est pair ou impair.

Suggestion : pour créer une fonction, il faut utiliser l'instruction « def », puis indiquer le nom de la fonction, avec ses paramètres (entrée)

```
def estPair(numero) :
```

Suggestion : pour retourner une valeur à partir d'une fonction, on utilise l'instruction « return ». Par exemple, pour retourner la valeur d'une variable « estPair » (sortie)

```
return estPair
```

- b) Modifier le code fait dans la question précédente pour utiliser cette fonction.
- c) Séparer la fonction du reste du code en plaçant le code de cette fonction dans un autre fichier (« FonctionEstPair.py »). Modifier le programme pour qu'il puisse importer cette fonction à partir du fichier que vous avez créé (« FonctionEstPair.py »).

Suggestion : pour utiliser une fonction, il faut d'abord l'importer dans le code.

```
from FonctionEstPair import estPair
```

TD B :

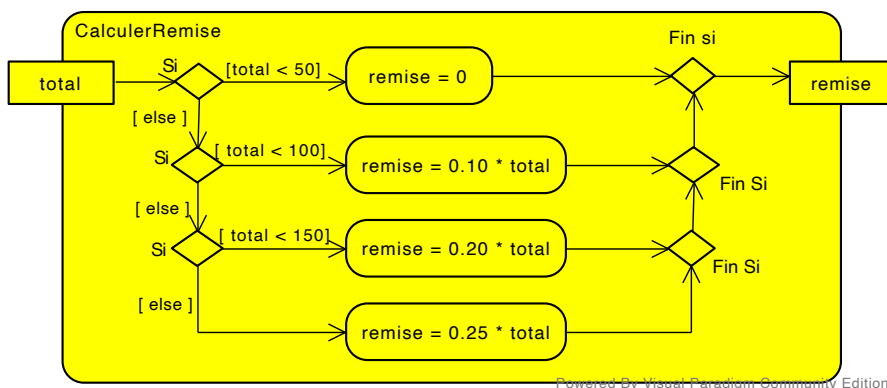
Exercice pas à pas :

Pour ce premier exercice, nous allons reprendre un exercice réalisé lors de la fiche n° 4 Algorithmes : l'exercice de la remise.

Dans une boutique de vêtements, on souhaite appliquer une remise de prix en fonction de la valeur total des achats. Pour plus de 50€ en achat, on souhaite offrir aux clients une remise de 10%. Pour plus de 100€ d'achat, on lui offrira une remise de 20%, alors que pour plus de 150€ d'achats, on offrira 25%. A l'aide d'un diagramme d'activité, proposez un algorithme permettant de calculer la valeur de la remise à partir d'un montant total d'achat donné.

Ce premier exercice représente un tutoriel, qui vous pouvez faire seul ou accompagné de votre chargé de TD. Lisez avec attention les instructions et suivez-les pas à pas.

Cet exercice, nous l'avons réalisé lors des fiches précédentes. Nous allons donc commencer rappeler l'algorithme qu'on avait alors proposé pour calculer la remise. Nous avons fait quelque chose ressemblant à l'algorithme ci-dessous :

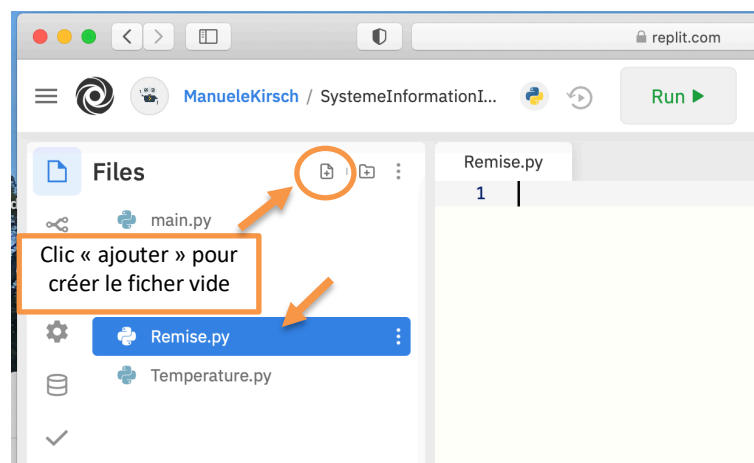


Dans celui-ci, on enchaîne une séquence de décisions (des « si »), construisant ainsi une structure de type « si – sinon si – sinon ». A chaque « si » on vérifie que notre achat se trouve (ou non) dans un certain palier. Au fil des conditions, on élimine petit à petit des paliers, pour retrouver celui qui correspond à l'achat. A

la fin, quel que soit le palier de remise choisi, la variable « remise » sera toujours définie avec une certaine valeur, calculée en fonction du total.

Maintenant que nous avons revu l'algorithme, nous pouvons passer à la traduction de celui-ci en Python. Avant de commencer, nous allons sur l'espace **Repl.it** qu'on a créé pour réaliser la première partie de cette fiche, et nous allons créer un nouveau fichier « **Remise.py** ».

Pour que cet algorithme fonctionne, il lui faut une valeur en **entrée**, le montant **total** des achats. Nous allons



```
Remise.py
1 # calcule de la remise
2
3 montant = input("montant ? ")
4 montant = float(montant)
5
6 print(montant)
```

donc commencer par **demander à l'utilisateur** de nous fournir cette valeur, à l'aide de la fonction « **input** ».

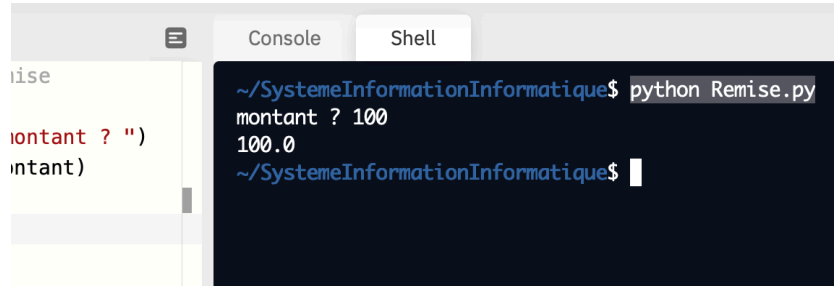
Cette fonction retourne une chaîne de caractères, on doit donc convertir cette chaîne dans un numéro réel (décimal), ce qui est possible à l'aide de la fonction « **float** ». On peut faire ça en deux étapes

comme ici (une ligne pour la fonction « input » et une ligne pour la conversion avec la fonction « float »), ou dans une seule ligne, comme on l'a fait dans la première partie de cette fiche.

```
montant = float (input ("montant ? "))
```

Pour tester ce que nous venons de faire, nous allons ajouter une ligne « `print(montant)` » pour afficher le montant qu'on vient de lire et nous allons exécuter le code sur l'onglet « `shell` » et taper :

```
python Remise.py
```



```
~/SystemeInformationInformatique$ python Remise.py
montant ? 100
100.0
~/SystemeInformationInformatique$
```

Maintenant que nous avons dans la variable « `montant` » notre valeur d'entrée, nous allons pouvoir traduire la structure de « `si – sinon si – sinon` » dans une structure « `if – elif – else` ». On n'oublie pas les « `:` » et le bon **respect des espaces** dans chaque bloc (Repl.it propose déjà le bon nombre d'espaces, il faut juste le respecter).

Une fois cette structure réalisée, nous avons dans la variable « `remise` » le montant de la remise à appliquer sur le montant d'achat.

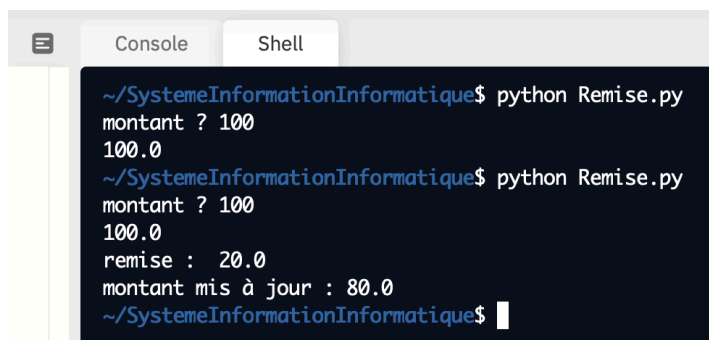
Nous devons donc mettre à jour ce montant d'achat (représenté par la variable « `montant` ») pour l'ajuster en fonction de la remise.

```
montant = montant - remise
```

```
8  if montant < 50 :
9  |   remise = 0
10 elif montant < 100 :
11 |   remise = montant * 0.10
12 elif montant < 150 :
13 |   remise = montant * 0.20
14 else :
15 |   remise = montant * 0.25
16
```

Une fois le montant d'achat ajusté, il nous ne reste qu'à l'afficher pour l'utilisateur à l'aide de la fonction « `print` », avant d'exécuter notre code.

```
print ('remise : ', remise)
print ('montant mis à jour : ' , montant)
```



```
~/SystemeInformationInformatique$ python Remise.py
montant ? 100
100.0
~/SystemeInformationInformatique$ python Remise.py
montant ? 100
100.0
remise : 20.0
montant mis à jour : 80.0
~/SystemeInformationInformatique$
```

Pensez à bien tester toutes les possibilités (un achat de moins de 50€, un achat entre 50€ et 100€, entre 100€ et 150€, un achat de plus de 150€) afin de vérifier que votre code fonctionne dans tous les cas.

Code complet :

```
# calcul de la remise

montant = input ("montant ? ")
montant = float (montant)

print(montant)

if montant < 50 :
    remise = 0
```



```
elif montant < 100 :
    remise = montant * 0.10
elif montant < 150 :
    remise = montant * 0.20
else :
    remise = montant * 0.25

montant = montant - remise

print ('remise : ', remise)
print ('montant mis à jour :' , montant)
```

Exercices :

- 1) A partir de l'étude de cas du fleuriste (étudié précédemment), réaliser un programme en Python capable de lire le nombre de fleur demandé dans un bouquet, le message qui l'accompagne (qui peut être vide) et de calculer le prix qui ça coûtera pour le client. *Suggestion : utiliser la fonction isspace. Par exemple, on peut faire pour une variable « lettre » :*

```
lettre.isspace()
```

Attention au dernier mot : afficher le nombre de mots comptabilisé pour être sûr qu'on l'a comptabilisé.

Le fleuriste « La fleur du coin » propose des bouquets à un prix compétitif par un modèle business assez simple : chaque fleur est facturée 4€. En plus des fleurs, on va également facturer la carte (lorsqu'un message accompagne le bouquet). Si le message contient plus de 10 mots, on facture 0,10€ par mot, sinon on facture 0,05€ par lettre. Pour calculer le prix du message, on va se servir de deux variables : nbMots et nbLettres. On va donc parcourir le message et analyser chaque caractère présent dans celui-ci. S'il s'agit d'un espace en blanc, la valeur de nbMots doit être augmentée ; sinon, il s'agit d'une lettre, on va alors mettre à jour la valeur de nbLettres. A la fin du processus, le montant total calculé pour le bouquet est affiché.

- 2) Développer une application Python (c.a.d. un programme) pour faire un jeu de devinette. L'utilisateur lui donne un chiffre, l'ordinateur doit tirer au sort un numéro entre 0 et ce chiffre, et l'utilisateur doit ensuite essayer de découvrir quel numéro l'ordinateur a tiré au sort. L'utilisateur a droit à 3 essais.

Suggestion : utiliser la fonction « randint (min, max) » proposée par la bibliothèque « random ». Cette fonction permet de tirer au sort un numéro entier entre min et max. Par contre, pensez à bien importer cette fonction, pour que l'interpréteur Python puisse la trouver.

```
from random import randint
```

Conseil : préparer d'abord l'algorithme que vous allez suivre.

3) Lors des fiches précédentes, nous avons préparé un algorithme pour inverser le contenu d'un tableau. Par exemple, un [1 , 2 , 3 , 4 , 5] devenait [5 , 4 , 3 , 2 , 1]. Nous allons maintenant implémenter différents algorithmes permettant de faire ceci.

a) Tout d'abord, construire un programme en Python permettant de renseigner un tableau de valeurs. On lui demandera combien des valeurs il souhaite pour son tableau puis on lui demandera autant de valeurs.

Suggestion : on peut créer un tableau de taille n grâce à la ligne ci-dessous (voir exemple « creationLists.py »)

```
liste = [ None for col in range(n) ]
```

b) Créer un deuxième tableau « listeInv ». A l'aide d'une boucle, récupérer les valeurs dans le tableau « liste » et placer-les en ordre inverse dans le tableau « listeInv ». Par exemple, si le tableau « liste » contient les valeurs « [1, 2, 3, 4, 5] », le tableau « listeInv » contiendra les valeurs « [5, 4, 3, 2, 1] ». A la fin imprimer les deux tableaux.

Suggestion : inspirez-vous des algorithmes qu'on a fait dans les fiches précédentes.

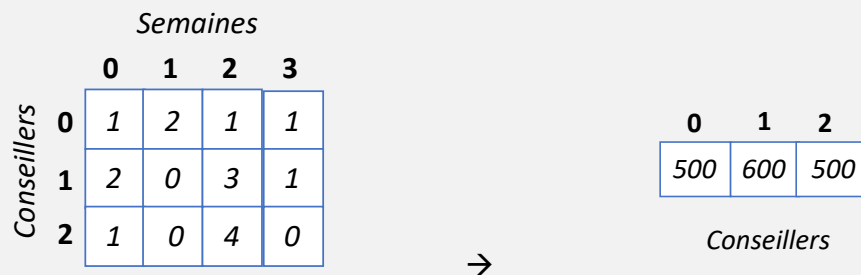
c) A l'aide d'une nouvelle variable « swap » (une variable de « sauvegarde »), faire une nouvelle boucle qui va inverser le contenu du tableau originel (« liste »). Si avant l'exécution de cette boucle, il contenait les valeurs « [1, 2, 3, 4, 5] », à la fin de celle-ci, il contiendra alors les valeurs dans l'ordre inversée « [5, 4, 3, 2, 1] ». Afficher le contenu du tableau.

Suggestion : inspirez-vous des algorithmes qu'on a fait dans les fiches précédentes. La variable de « sauvegarde » permet de mettre de côté une valeur du tableau pour l'échanger avec une autre.

d) Utiliser maintenant sur le tableau « listeInv » la méthode « reverse » et afficher le résultat (le tableau une fois la méthode exécutée). Que fait-elle ?

4) A partir de l'algorithme préparé dans les fiches précédentes, proposer un programme en Python pour calculer le montant des commissions pour les consultants de la banque « MySous ». Procéder suivant les étapes indiquées ci-dessous :

Dans la banque « MySous », les conseillers sont commissionnés. A chaque devis analysé, ils reçoivent 100€ de commission. Pour calculer le montant total de commission à payer pour chaque conseiller, on considère un tableau avec le nombre de devis analysés par conseiller par semaine (voir figure ci-dessous), et on affiche, en sortie, un tableau avec le montant de la commission pour chaque conseiller (voir figure ci-dessous).



- a) Le programme doit d'abord demander à l'utilisateur combien de conseillers il a à traiter. Puis pour chaque conseiller, le programme va demander le nombre de devis élaborés par semaine. Afficher l'ensemble de données récoltées.

Rappel : dans Python, un « tableau » (liste) est en effet un « tableau de tableaux » (c.a.d. un tableau est une liste dont chaque position est une autre liste). Dans notre cas, ça veut dire que chaque ligne (représentant un conseiller) est un tableau de 4 positions (pour les 4 semaines qui existent dans un mois)

Suggestion : pensez d'abord l'algorithme qu'il vous vaudra pour renseigner ces valeurs.

- b) A l'aide de l'algorithme préparé dans les fiches précédentes, créer un nouveau tableau pour enregistrer les montant de commissions et calculer ces commissions.

- 5) Améliorer le programme qui vous venez de faire à l'exercice 4), à l'aide de la notion de fonctions. On va créer une fonction « calculCommission » qui reçoit à l'entrée un tableau de 2 dimensions (nb de conseillers x nb de semaines) avec le nombre de devis réalisés par semaine pour chaque conseiller, et qui retourne en sortie, un tableau avec les montants de commissions calculées par conseiller.

- a) Placer le code que vous avez créé dans la question 4)b) sur une fonction « calculCommission ». Placer celle-ci dans un autre fichier « FonctionsCommissions » et **adapter** le code pour qu'il fonctionne correctement.

Rappel : pour créer une fonction, il faut utiliser l'instruction « def » et indiquer le nom de la fonction avec ses paramètres (entrée), et pour retourner une variable (sortie), il faut utiliser l'instruction « return » avec le nom de la variable :

```
def nomFonction(parametre) :  
    # code (contenu) de la fonction  
    return variableSortie
```

- b) Remplacer le code de la question 4)b) par un appel à la fonction « calculCommission » que vous venez de créer.

Rappel : pour utiliser une fonction, il faut d'abord l'importer dans le code. Une fois importée, on peut l'utiliser (l'invoquer) autant de fois qu'on le souhaite.

```
from FichierAvecLaFonction import nomFonction  
  
# code ...  
variable = nomFonction (parametre)
```

- 6) Lors des fiches précédentes, nous avons réalisé un algorithme permettant d'inverser le contenu d'une matrice M, ou plutôt *transposer le contenu de la matrice M* (voir figure ci-dessous). Nous allons maintenant explorer différentes manières d'implémenter cet algorithme. Pour cela, on va procéder étape par étape :



- a) Tout d'abord, nous allons nous occuper de la création de la matrice M. Nous allons demander à l'utilisateur le nombre de ligne et de colonnes qu'il souhaite pour sa matrice, puis, à l'aides des boucles, nous allons lui permettre de remplir la matrice.

Suggestion : inspirez-vous de ce que vous avez fait dans la question 4)a).

- b) Ensuite, nous allons mettre en œuvre l'algorithme que nous avons préparé lors des fiches précédentes.

Suggestion : pour créer un tableau de 2 dimensions L x C, on peut se servir de ligne suivante (plus d'informations dans l'exemple « creationLists.py »).

```
tableau = [ [ 0 ] * C for i in range(L) ]
```

- c) Enfin, dans un nouveau code, nous allons réaliser la même chose, mais en utilisant la bibliothèque NumPy. Celle-ci de créer facilement des matrices de dimensions (L, C) et propose des nombreuses méthodes pour leur manipulation.

Attention : si vous utilisez Idle ou Visual Studio Code, regardez avant le tutoriel « Usage Bibliothèques » sur l'EPI.

On va devoir importer la bibliothèque NumPy dans notre code :

```
import numpy as np
```

Pour créer une matrice de taille (L, C) en NumPy, on va alors utiliser le code :

```
m = np.zeros( ( L, C ) )
```

pour créer une matrice m remplie des valeurs 0.0 ; ou encore

```
m = np.empty( (nbLignes,nbColonnes), dtype='object' )
```

pour créer une matrice m vide (remplie de valeurs 'None').

Pour transposer une matrice m, on peut utiliser la méthode « transposer » :

```
mInv = m.transpose()
```