Correction de quelques exercices de la feuille de TD 3

Exercice 1 (Opérations sur les nombres).

1) a) Exécuter les commandes suivantes dans la console en essayant de comprendre (voire même d'anticiper) leur résultat :
   - `2+5` `2*5` `2/5` `5,2`
   - `2-5` `2**5` `5.2` `5;2`

   b) Quelle est la différence entre le point et la virgule ? Que fait le point virgule ?
2) Appuyer plusieurs fois sur les touches fléchées haut/bas du clavier ? En quoi cela va nous être très utile ?
3) a) Quel est l'ordre de priorité des opérations en Mathématiques ?
   b) Comment faire des barres de fractions horizontale ou la notation en exposant avec Python ?
   c) Calculer, à l'aide de Python, des valeurs approchée de
   - $87 \times 2021 - 21$
   - $4^{5/3}$
   - $-7 + \dfrac{4}{7 - \frac{4+7}{4\times 7}}$
   - 3^{2^5}
   - $\dfrac{2021}{8 \times 29}$
4) Exécuter les commandes suivantes dans la console en essayant de comprendre (voire même d'anticiper) leur résultat :
   - `22/7` `22//7` `22%7` `int(22/7)`
   - `355/113` `355//113` `355%113` `int(-355/113)`

   Que font les commandes `//`, `%` et `int` ?

Correction :

1) a) Successivement :
   - `7` `10` `0.4` `(5,2)`
   - `-3` `32` `5.2` `2`

   b) Le point est le séparateur entre les chiffres des unités et la partie décimale d'un réel. La commande `x,y` créer le couple formé de `x` et `y`. Le point virgule est juste un séparateur entre deux instructions (en général seule le résultat de la dernière s'affiche)... pratique pour faire plusieurs instruction sur une ligne.
2) Cela réaffiche les commandes précédemment exécutées. Cela évite de recopier ou même de faire des copier/coller.
3) a) PEMDAS... cf. cours.
   b) On ne peut pas... il faut faire un usage adéquat des parenthèses.
   c) Il faut taper respectivement :
   - `87*201-21`
   - `4**(5/3)`
   - `-7+4/(7-(4+7)/(4*7))`
   - `3**(2**5)`
   - `2021/(8*29)`
4) Successivement :
   - `3.142857142857143` `3` `1` `3`
   - `3.1415929203539825` `3` `16` `-3`

   Cf. cours pour l'explication de `//`,`%`,`int`. Attention `int` n'est pas la partie entière (seulement si on lui donne un réel positif).

Exercice 2 (Création, affectation, manipulation de variables).

1) a) Exécuter les commandes suivantes dans la console en essayant de comprendre (voire même d'anticiper) leur résultat :

• `x`	• `7=x`	• `x`	• `x=x*5`	• `x`
• `x=7`	• `x==7`	• `x=x+2`	• `x`	• `x=x**(1/2)`
• `x`	• `x+2`	• `x`	• `x/3`	• `x`

b) Quelle est la différence entre = et == ?

c) Comment affecte-t-on une valeur dans une variable ? Comment mettre à jour le contenu d'une variable ?

2) Exécuter les commandes suivantes dans la console en essayant d'anticiper leur résultat :

• `x=9`	• `x=y-x`	• `z=z*z`	• `res2=4*res+8`
• `y=2`	• `z=x**y/7`	• `res=z/y+3.5`	• `res3=res2/10`

3) Comment échanger le contenu des variables `x` et `y` (sans savoir ce qu'elles contiennent) ? Essayer avec `x=9;y=2`.

4) Que fait la commande `x+=3` ? Et la commande `x*=5` ?

Correction :

1) a) Successivement :
   - Message d'erreur (`x` n'est pas défini)
   - Affecte 77 dans la variable `x`
   - Affiche le contenu de la variable `x`, c'est-à-dire 7.
   - Message d'erreur (ça n'a aucun sens !)
   - Teste si `x` est égal à 7. C'est le cas donc la commande renvoie `True`.
   - Affiche le résultat de `x+2`, c'est-à-dire 9.
   - Affiche le contenu de la variable `x`, c'est-à-dire 7.
   - Effectue le calcul `x+2`, ce qui donne 9, puis affecte le résultat de ce calcul dans `x` (Python oublie que `x` valait 7. Désormais `x` contient 9).
   - Affiche le contenu de la variable `x`, c'est-à-dire 9.
   - Effectue le calcul `x*5`, ce qui donne 45, puis affecte le résultat de ce calcul dans `x` (Python oublie que `x` valait 9. Désormais `x` contient 45).
   - Affiche le contenu de la variable `x`, c'est-à-dire 45.
   - Effectue le calcul `x/3`, ce qui donne 15.
   - Affiche le contenu de la variable `x`, c'est-à-dire 45.
   - Effectue le calcul `x**(1/2)`, ce qui donne `6.708203932499369` (une valeur approchée de $\sqrt{45}$), puis affecte le résultat de ce calcul dans `x` (Python oublie que `x` valait 45. Désormais `x` contient `6.708203932499369`). `x=x**(1/2)`
   - Affiche le contenu de la variable `x`, c'est-à-dire `6.708203932499369`.

   b) = sert à affecter une valeur dans une variable tandis que == teste si les deux valeurs de part et d'autres sont les mêmes (elle renvoie `True` ou `False`).

   c) Cf. cours.

2) Successivement :
   - `x` contient 9.
   - `y` contient 2.
   - `x` contient maintenant le résultat de `y-x`, c'est-à-dire 7.
   - `z` contient le résultat de `x**y/7`, c'est-à-dire `7.0` (car $7^2/7 = 7$).
   - `z` contient maintenant le résultat de `z*z`, c'est-à-dire `49.0`.

- `res` contient le résultat de `z/y+3.5`, c'est-à-dire `28.0` (car $49/2 + 3,5 = 24,5 + 3,5 = 28$).
- `res2` contient le résultat de `4*res+8`, c'est-à-dire `120.0` (car $4 * 28 + 8 = 120$).
- `res3` contient le résultat de `res2/10`, c'est-à-dire `12.0`.

Notons que le résultat final est un flottant (bien qu'entier mathématique) puisque l'on a fait des opérations non entières pour l'obtenir.

3) Si on écrit `x=y`, on met le contenu de `y` dans `x` mais on perd à tout jamais le contenu de `x`. Deux possibilités pour s'en sortir :
   - On utilise une variable auxiliaire : `aux=x; x=y; y=aux`.
   - On passe par des `tuples` (des couples Python) : `x,y=y,x`.
4) `x+=3` ajoute 3 à `x` et stocke le résultat dans `x`. Cette commande fait la même chose que `x=x+3`.
   `x*=5` multiplie `x` par 5 et stocke le résultat dans `x`. Cette commande fait la même chose que `x=x*5`.

Exercice 3 (Variables booléennes).

1) Exécuter les commandes suivantes dans la console en essayant d'anticiper leur résultat :
   - `0=1`
   - `0!=1`
   - `0<=1`
   - `0>=1`
   - `y=(0<1)`
   - `0==1`
   - `0<1`
   - `0>1`
   - `x=(0==1)`
   - `x==y`
2) Exécuter les commandes suivantes dans la console en essayant d'anticiper leur résultat :
   - `True or True`
   - `True and False`
   - `4>1 and 4<-1`
   - `True or False`
   - `False and False`
   - `4>1 or 4<-1`
   - `True and True`
   - `not False`
   - `(not 0==1) and (2>1)`
3) Soient `x` et `y` deux variables qui contiennent des booléens (`True` ou `False`). Écrire une commande qui renvoie `True` si `x` est vraie mais pas `y` ou si `y` est vraie mais pas `x` et qui renvoie `False` si toutes les deux sont fausses ou toutes les deux sont vraies.
4) Soit `x` une variable contenant un réel.
   a) Quelle commande permet de tester si ce réel appartient à l'intervalle $[3\,;19[$?
   b) Quelle commande permet de tester si ce réel appartient à $]-\infty\,;12] \cup]293\,;765[$?
5) Soit `x` une variable contenant un réel.
   a) Quelle commande permet de tester si ce réel est un nombre entier ?
   b) Désormais on sait que `x` contient un nombre entier. Quelle commande permet de tester s'il est pair ?
   c) Quelle commande permet de tester si cet entier est impair ?
   d) Quelle commande permet de tester si cet entier est un multiple de 7 ?

Correction :

1) Successivement :
   - Message d'erreur (on ne peut pas stocker `1` dans `0` allons !).
   - `0==1`
   - `False`
   - `True`
   - `True`
   - `False`
   - `False`
   - On stocke le résultat de `0==1` (c'est-à-dire `False`) dans `x`.
   - On stocke le résultat de `0<1` (c'est-à-dire `True`) dans `y`. `y=(0<1)`
   - Teste si `x` et `y` contiennent le même booléen. Ce n'est pas le cas donc renvoie `False`.

2) Successivement
   - `True`
   - `True`
   - `True`
   - `False`
   - `False`
   - `True`
   - `False`
   - `True`
   - `True`

3) Première possibilité : `(x==True and y==False) or (x==False and y==True)`
   Deuxième possibilité : `(x and not(y)) or (not(x) and y)`
   Troisième possibilité : `not(x==y)`
   Quatrième possibilité : `x!=y`

4) a) `3<=x<19`
   b) `(x<=12) or (293<x<765)`

5) a) `int(x)==x`
   b) `x%2==0` ou encore `int(x/2)==x/2`
   c) `x%2==1` ou encore `int(x/2)!=x/2`
   d) `x%7==0` ou encore `int(x/7)==x/7`

Exercice 4 (Chaînes de caractères).

1) Exécuter les commandes suivantes dans la console :
   - `Bonjour`
   - `"Bonjour"`
   - `'Bonjour'`
   - `x='Bonjour'; x`
   - `y='Je m'appelle Bond, James Bond.'`
   - `y='Je m''appelle Bond, James Bond.'`
   - `x+y`
   - `z=x+". "+y`

2) Exécuter les commandes suivantes dans la console :
   - `a=2; b="2"; c=5; d="5"`
   - `a+d`
   - `a+c`
   - `b+d`

3) Exécuter les commandes suivantes dans la console :
   - `z + " " + "Mon matricule est 00" + str(a+c) + "."`
   - `x=355/113; "Une approximation de pi est x."`
   - `"Une approximation de pi est " + str(x) + "."`

Correction :

1) Successivement :
   - Message d'erreur : la variable `Bonjour` n'est pas définie.
   - Écrit la phrase `Bonjour`.
   - Écrit la phrase `Bonjour`.
   - Stocke la phrase `Bonjour` dans la variable `x` et l'affiche.
   - Message d'erreur car veut stocker la phrase `Je m` dans la variable `y` et ne comprend pas ce que la commande `appelle Bond, James Bond.'` vient faire ici.
   - Stocke dans `y` la phrase `Je m'appelle Bond, James Bond.`
   - Concatène/fusionne les phrases contenues dans `x` et `y`. Cela affiche donc `Bonjour.Je m'appelle Bond, James Bond.`
     Notons qu'aucune espace entre les deux phrases ne se crée spontanément.
   - Concatène la phrase contenue dans `x`, la phrase `. ` (un point et un espace) puis la phrase contenue dans `y` et stocke le résultat dans `z`. Ainsi `z` contient la chaîne de caractère `Bonjour. Je m'appelle Bond, James Bond.`

2) Successivement :
   - Stocke l'entier 2 dans la variable `a`, stocke la phrase 2 dans la variable `b`, stocke l'entier 5 dans la variable `c`, stocke la phrase 5 dans la variable `d`.
   - Message d'erreur : cela n'a aucun sens d'ajouter un entier et une chaine de caractère.
     `a+d`
   - Additionne les entiers contenus dans `a` et `c` et affiche donc 7.
   - Concatène les phrases contenus dans `b` et `d` et affiche donc la phrase 25.
3) Successivement :
   - Affiche la phrase `Bonjour. Je m'appelle Bond, James Bond. Mon matricule est 007.`
   - Stocke `3.1415929203539825` dans `x` (c'est une valeur approchée de 355/113) puis affiche la phrase `Une approximation de pi est x.`
   - Affiche la phrase `Une approximation de pi est 3.1415929203539825.`

Exercice 5 (C'est l'histoire d'un type).

1) Exécuter les commandes suivantes dans la console :
   - `type(7)`
   - `type(22/7)`
   - `type(0==1)`
   - `type("Hello")`
2) Exécuter les commandes suivantes dans la console :
   - `type(7)`
   - `type(14/2)`
   - `type(7.0)`
   - `type(14//2)`
   - `7==14/2`
   - `7==7.0`
   - `7==7.00000001`
   - `7==7.0000000000000001`
   - `x=int(14/2)`
   - `type(x)`
   - `y=float(7)`
   - `type(y)`
3) Quelle est la différence entre le type `float` et le type `int` ?
4) Comment obtenir facilement la partie entière d'un nombre en Python ?

Correction :

1) Successivement :
   - `<class 'int'>` : C'est un entier Python.
   - `<class 'float'>` : C'est un flottant.
   - `<class 'bool'>` : C'est un booléen.
   - `<class 'str'>` : C'est une chaîne de caractère.
2) Successivement :
   - `<class 'int'>`
   - `<class 'float'>`
   - `<class 'float'>`
   - `<class 'int'>`
   - `True`
   - `True`
   - `False`
   - `True`
   - Stocke 7 dans `x`
   - `<class 'int'>`
   - Stocke `7.0` dans `x`
   - `<class 'float'>`

   Le huitième est curieux... en fait la précision Python par défaut est telle qu'il ne voit plus la différence entre les deux nombres au delà d'un certain nombre de chiffres après la virgule (peut donner `False` selon le logiciel employé et/ou la précision).
3) Cf. cours.
4) Si `x` implémente un réel, on peut utilise `int(x)` s'il est positif et `int(x)-1`, s'il est négatif.

Exercice 6 (Découverte des listes). Cet exercice est à traiter avec un ordinateur.

1) Exécuter dans la console la commande `L=[7,22/7,'Hello !',3==8,x+8,3.1**(1/2)]` puis les suivantes :
   - `L`
   - `type(L)`
   - `L[0]`
   - `L[1]`
   - `L[3]`
   - `L[-1]`
   - `L[-2]`
   - `len(L)`
   - `7 in L`
   - `5 in L`
   - `L[:3]`
   - `L[2:]`
   - `L[1:4]`
   - `Lc=L`
   - `Ls=L[:]`
   - `L, Lc, Ls`

2) Exécuter les commandes suivantes dans la console :
   - `L[1]='H1B'`
   - `L, Lc, Ls`
   - `L.append(17432)`
   - `L, Lc, Ls`
   - `L+[2,-6,'OK']`
   - `L`
   - `L=L[:3]+L[4:]`
   - `L`
   - `L.pop()`
   - `M=[7,18,[2,5]]`
   - `M[0]`
   - `M[2][1]`

3) Exécuter les commandes suivantes dans la console :
   - `range(7)`
   - `type(range(7))`
   - `[k for k in range(7)]`
   - `[k**2 for k in range(1,9)]`
   - `[i for i in range(-5,17)]`
   - `[j for j in range(-5,-19)]`

Correction :

1) Successivement :
   - `[7,3.142857142857143,'Hello !',False,15,1.760681686165901]`
   - `<class 'list'>`
   - `7`
   - `3.142857142857143`
   - `False`
   - `1.760681686165901`
   - `15`
   - `6`
   - `True`
   - `False`
   - `[7, 3.142857142857143, 'Hello !']`
   - `['Hello !', False, 15, 1.760681686165901]`
   - `[3.142857142857143, 'Hello !', False]`
   - Stocke le contenu de la liste `L` dans `Lc`... mais pour Python c'est le même object (tout changement sur l'un, provoque le même changement sur l'autre). `Lc=L`
   - Stocke le contenu de la liste `L` dans `Lc` mais en effectuant une copie : pour Python c'est pas le même object (tout changement sur l'un laisse inchangé le deuxième).
   - Affiche les trois listes. Elles sont identiques à ce stade.

2) Successivement :
   - Remplace élement d'indice `1` (le deuxième donc) de la liste `L` (et donc aussi celui de la liste `Lc`) par la chaîne de caractère `'H1B'`.
   - Affiche les trois listes. On voit que `Ls` n'a pas été modifiée.
   - Ajoute à la liste `L` (et donc aussi à la liste `Lc`) l'entier `17432` en dernière position.
   - Affiche les trois listes. On voit que `Ls` n'a pas été modifiée.
   - Créer une liste qui est la concaténation de `L` et de la liste `[2,-6,'OK']`
   - Affiche la liste `L` (elle n'a pas été modifié par la toute dernière instruction).

- Fusionne la liste `L[:3]` (elle est constituée des termes d'indices 0 à 2 de `L`) avec la liste `L[4,:]` (elle est constituée des termes d'indices supérieurs ou égaux à 4 de `L`). Elle stocke le résultat dans la liste `L`. Ainsi toute cette commande supprime de la liste `L` l'élément d'indice 3.
- Affiche la liste `L`. On voit que l'élément d'indice 3 a bien été supprimé.
- Retire de la liste `L` son tout dernier élément et l'affiche (il s'agit de17432).
- Créer la liste `[7,18,[2,5]]` et la stocke dans `M`.
- Affiche le terme d'indice 0 de `M`, il s'agit de 7.
- La commande `M[2]` affiche le terme d'indice 2 de `M`, il s'agit à son tour d'une liste (la liste `[2,5]`. Ainsi la commande `M[2][1]` affiche l'élément d'indice 1 de cette dernière liste, c'est-à-dire 5.

3) Successivement :
- `range(0, 7)`
- `<class 'range'>`
- `[0,1,2,3,4,5,6]`
- `[1,4,9,16,25,36,49,64]`
- `[-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]`
- `[]`

Exercice 10. Implémenter en Python la fonction f qui est 2-périodique sur $\mathbb{R}$ et qui vérifie

$$\forall x \in [-1\,;1[\,, \qquad f(x) = x\sqrt{1-x^2}.$$

On pourra commencer par exprimer, en fonction de $x \in \mathbb{R}$, l'unique entier k qui vérifie $-1 < x - 2k \leqslant 1$.

Correction : Soit $x \in \mathbb{R}$. On a $-1 \leqslant x - 2k < 1$ si et seulement si $-1 + 2k \leqslant x < 1 + 2k$ si et seulement si $k \leqslant \dfrac{x+1}{2} < k+1$ si et seulement si $k = \left\lfloor \dfrac{x+1}{2} \right\rfloor$. On utilise le fait que $f(x) = f(x-2k)$:

```
def f(x):
    k=int((x+1)/2)
    y=x-2*k
    return y*(1-y**2)**(1/2)
```

Exercice 13. Écrire une fonction python dont l'argument est une année et qui renvoie un booléen selon que l'année est bissextile ou non.

Une année est bissextile (elle a 366 jours), si elle est divisible par 4 et non divisible par 100, ou si elle est divisible par 400.

Correction :

```
def EstBissextile(n):
    if n%4==0 and n%100!=0 and n%400==0:
        return True
    else:
        return False
```

ou plus simplement :

```
def EstBissextile(n):
    return n%4==0 and n%100!=0 and n%400==0:
```

Exercice 15. Écrire une fonction qui prend en argument quatre réels a, b et c et qui affiche une phrase indiquant l'ensemble des réels x tels que $ax^2 + bx + c > 0$.

On utilisera les commandes `print` *et* `str` *(cette dernière convertit un réel en chaîne de caractères).*

Correction : *Exercice difficile à ce stade de l'année... mais il devra vous sembler simple au plus vite !*

Si $a = 0$, on cherche x tel que $bx + c > 0$, c'est-à-dire $bx > -c$.

- Si $b > 0$, c'est équivalent à $x > -c/b$.
- Si $b < 0$, c'est équivalent à $x < -c/b$.
- Si $b = 0$, c'est équivalent à $0 > -c$ et donc à $c > 0$. Le trinôme est alors strictement positif TOUT LE TEMPS lorsque $c > 0$ et JAMAIS si $c \leqslant 0$.

Si $a \neq 0$, il s'agit de déterminer le signe d'un trinôme du second degré. Notons Δ son discriminant.

- Si $\Delta > 0$, le trinôme est strictement positif lorsque :
  — x est strictement « entre les racines » dans le cas où $a < 0$.
  — x est strictement « en dehors des racines » dans le cas où $a > 0$.
- Si $\Delta = 0$, le trinôme est strictement positif :
  — JAMAIS si $a < 0$.
  — TOUT LE TEMPS sauf quand x est égal à l'unique racine, $-b/(2a)$.
- Si $\Delta > 0$, , le trinôme est strictement positif :
  — JAMAIS si $a < 0$.
  — TOUT LE TEMPS si $a > 0$.

```
def TrinomePositif(a,b,c):
    if a==0:
        if b>0:
            print(']'+str(-c/b)+',+infini[')
        elif b<0:
            print(']-infini,'+str(-c/b)+'[')
        else:
            if c>0:
             print(']-infini,+infini[')
            else:
                print('vide')
    else:
        D=b**2-4*a*c
        if D>0 and a>0:
            x1=(-b-np.sqrt(D))/(2*a);
            x2=(-b+np.sqrt(D))/(2*a);
            print(']-infini,'+str(x1)+'[ U ]'+str(x2)+',+infini[')
        elif D>0 and a<0:
            x1=(-b-np.sqrt(D))/(2*a);
            x2=(-b+np.sqrt(D))/(2*a);
            print(']'+str(x2)+','+str(x1)+'[')
        elif D==0 and a>0:
              x0=-b/(2*a);
              print(']-infini,'+str(x0)+'[ U ]'+str(x0)+',+infini[')
        elif D<0 and a>0:
              print(']-infini,+infini[')
        else:
            print('vide')
```

Exercice 16. Implémenter en Python la fonction $f : x \longmapsto \sqrt[4]{1 - \dfrac{5}{3+x}}$.

Un message d'erreur sera affiché lorsqu'on évalue f en un réel en lequel elle n'est pas définie.

Correction : On a $D_f = \left\{ x \in \mathbb{R} \,\middle|\, 3 + x \neq 0 \quad \text{et} \quad 1 - \dfrac{5}{3+x} \geqslant 0 \right\}$. Soit $x \in \mathbb{R}\backslash\{-3\}$. On a $1 - \dfrac{5}{3+x} \geqslant 0$ si et seulement si $1 \geqslant \dfrac{5}{3+x}$. Pour inverser, tout doit avoir le même signe :

- Si $x < -3$, alors $3 + x < 0$ donc $\dfrac{3}{5+x} < 0$ donc $\dfrac{3}{5+x} \leqslant 1$ et donc $1 - \dfrac{5}{3+x} \geqslant 0$.
- Si $x > -3$, alors $3 + x > 0$ donc $1 \geqslant \dfrac{5}{3+x}$ si et seulement si $3 + x \geqslant 5$ si et seulement si $x \geqslant 2$.

Ainsi $D_f =]-\infty\,;-3[\cup [2\,;+\infty]$.

```
def f(x):
    if x<-3 or x>=2:
        return (1-5/(3+x))**(1/4)
    else:
        print('erreur')
```

Exercice 17. Écrire une fonction Python sans argument qui, lorsqu'on l'exécute, simule la saisie du code confidentiel d'une carte bancaire : l'utilisateur a trois essais pour taper le bon code et ensuite la carte est bloquée (on peut prendre par exemple 2023 comme code confidentiel).

Indication : la commande `a=int(input("texte"))` *affiche la chaine de caractère* `texte`. *L'utilisateur tape alors un entier dans la console et cette entier sera stocké dans la variable* `a`.

Correction :

```
def Jeu():
    code=2023;#Le vrai code (connu ou pas de l'utilisateur)
    n=int(input('Entrez votre code confidentiel :'))
    if n==code:
        print('Code bon')
    else:
        n=int(input('Code erroné. Deuxième essai :'))
        if n==code:
            print('Code bon')
        else:
            n=int(input('Code erroné. Dernier essai :'))
            if n==code:
                print('Code bon')
            else:
                print('Code erroné. Carte avalée.')
```

Exercice 24 (Recherche du max). Écrire une fonction qui prend en argument une liste de réels et qui renvoie le maximum de la liste (sans utiliser `np.max`), ainsi que le plus petit indice où se trouve le maximum.

Indications : le premier élément de la liste est le maximum provisoire. Ensuite on parcourt la liste de gauche à droite et, dès qu'on trouve un élément strictement plus grand que le maximum provisoire, celui-ci devient le nouveau maximum provisoire.

Correction :

```
def Ou_est_le_max(L):
    imax=0#Au début de l'exploration, le max est en indice 0
    n=len(L)
    for i in range(1,n):
        if L[i]>L[imax]:
            imax=i
    return [L[imax],imax]
```

Exercice 25 (Le juste prix). Écrire une fonction qui prend en argument un entier naturel n, qui choisit au hasard un entier entre 1 et n et qui demande à l'utilisateur de deviner le nombre choisi et qui répond *plus* ou *moins* selon la valeur proposée par l'utilisateur. Le programme continue d'interroger l'utilisateur jusqu'à ce qu'il devine la bonne réponse.

Indications :

- On exécute la commande `import numpy.random as rd`. *Ensuite la commande* `rd.randint(1,n+1)` *renvoie un entier tiré au hasard entre* 1 *et* n. *On y reviendra plus en détail lors des TP de Probabilités...*
- *La commande* `a=int(input("texte"))` *affiche la chaine de caractères* `texte`. *L'utilisateur tape alors un entier dans la console et cette entier sera stocké dans la variable* `a`.

Correction :

```
import numpy.random as rd

def JustePrix(n):
    print('jouons! Je vais choisir un entier entre 1 et '+str(n)+' et
    vous devez le deviner.')
    x=rd.randint(1,n+1)
    c=0#Un compteur
    v=int(input('Devinez le nombre que j''ai choisi :'))
    while x!=a:
          if int(x)!=x:
             print('Ce n''est pas un entier allons. Recommencez.')
          elif x>a:
             v=int(input('C''est moins! Recommencez :'))
          else:
             v=int(input('C''est plus! Recommencez :'))
          c=c+1
    print('Bravo! Tu as trouvé le nombre en '+str(c)+' essais.')
```

Exercice 26. Le nombre 4150 possède une propriété remarquable : il est égal à la somme des puissances cinquièmes de ses chiffres. En effet

$$4150 = 4^5 + 1^5 + 5^5 + 0^5.$$

1) En raisonnant par l'absurde, montrer que si un nombre vérifiant cette propriété a au plus 6 chiffres.
2) En déduire que seul un nombre fini d'entiers vérifie cette propriété.
3) Écrire un programme Python qui affiche les nombres satisfaisant cette propriété.

Correction :

1) Considérons un tel nombre N qui possède $n \geqslant 7$ chiffres. Ce nombre s'écrit donc sous la forme $i_n i_{n-1} \cdots i_3 i_2 i_1$ où $i_k \in \{0;9\}$ désigne le $k^{\text{ième}}$ (en partant de la droite) chiffre de l'écriture de n du nombre (en base 10). On a donc

$$N = i_1^5 + i_2^5 + i_3^5 + \cdots + i_{n-1}^5 + i_n^5 \leqslant 9^5 + 9^5 + 9^5 + \cdots + 9^5 + 9^5 = n \times 9^5.$$

Ainsi $N \leqslant 7 \times 9^5 = 413343$ et donc N possède au plus 6 chiffres : c'est absurde ! On en déduit que N possède au plus 6 chiffres.

2) Ainsi, il y au plus 1000000 (on n'oublie pas 0) nombres vérifiant cette propriété. En particulier il y en a un nombre fini.

3) On va parcourir tous les entiers de 0 à 999999 en faisant 6 boucles dont chacune parcourt un chiffre de 0 à 9 de son écriture décimale. On teste si la somme des puissances cinquième du nombre lui est égal.

```
L=[];
for a in range(10):
    for b in range(10):
        for c in range(10):
            for d in range(10):
                for e in range(10):
                    for f in range(10):
                        n=f+e*10+d*100+c*1000+b*10000+a*100000;
                        if a**5+b**5+c**5+d**5+e**5+f**5==n:
                          L=L+[n]
print(L)
```

Exercice 28.

1) Que fait la fonction suivante ?

```
def mystere1(L):#L est une liste
    M=L[:]#On copie la liste L pour ne pas la modifier
    n=len(L)
    for i in range(n-1):
        imin=i
        for k in range(i+1,n):
            if M[k]<M[imin]:
                imin=k
        M[i],M[imin]=M[imin],M[i]
    return M
```

2) Et la suivante ?

```
def mystere2(L):#L est une liste
    M=L[:]#On copie la liste L pour ne pas la modifier
    n=len(M)
    for i in range(1,n):
        k=i
        while k>0 and M[k]<M[k-1]:
            M[k],M[k-1]=M[k-1],M[k]
            k=k-1
    return M
```

Correction : A VENIR