# Devoir d'informatique n° 01 (2 heures)

Ce devoir est constitué de trois exercices .
L'ordre des exercices ne correspond à aucun critère de difficulté ou de longueur : vous pouvez les traiter dans l'ordre que vous voulez.
Veillez à soigner la copie tant pour l'écriture, la propreté que pour la rédaction, la rigueur et l'argumentation. Vous numéroterez vos copies et ferez apparaître clairement sur la première page le nombre de copies.
La calculatrice n'est pas autorisée.

# EXERCICE I : Valuation $p-$adique

Dans cet exercice, on se propose d'écrire un algorithme pour décomposer un entier en produit de nombres premiers.
Les algorithmes demandés doivent être écrits en langage **Python**. On sera très attentif à la rédaction et notamment à l'indentation du code.

On définit la valuation $p-$adique d'un entier naturel non nul $n$ avec $p$ un nombre premier.

Si $p$ divise $n$, on note $v_p(n)$ le plus grand entier $k$ tel que $p^k$ divise $n$.

Si $p$ ne divise pas $n$, on pose $v_p(n) = 0$.

L'entier $v_p(n)$ s'appelle la valuation $p-$adique de $n$.

**1.** Écrire une fonction booléenne `estPremier(n)` qui prend en argument un entier naturel non nul $n$ et qui renvoie le booléen `True` si $n$ est premier et le booléen `False` sinon. On pourra utiliser le critère suivant : un entier $n \geqslant 2$ qui n'est divisible par aucun entier $d \geqslant 2$ tel que $d^2 \leqslant n$, est premier.

**2.** En déduire une fonction `liste_premier(n)` qui prend en argument un entier naturel non nul $n$ et renvoie la liste des nombres premiers inférieurs ou égaux à $n$.

**3.** Pour calculer la valuation $2-$adique de 40, on peut utiliser la méthode suivante :

- ✏ 40 est divisible par 2 et le quotient vaut 20
- ✏ 20 est divisible par 2 et le quotient vaut 10
- ✏ 10 est divisible par 2 et le quotient vaut 5
- ✏ 5 n'est pas divisible par 2.

La valuation $2-$adique de 40 vaut donc 3.

Écrire une fonction `val(n, p)` qui implémente cet algorithme. Elle prend en argument un entier naturel $n$ non nul et un nombre premier $p$, et renvoie la valuation $p-$adique de $n$. Par exemple, puisque $40 = 2^3 \times 5$, `val(40, 2)` renvoie 3, `val(40, 5)` renvoie 1 et `val(40, 7)` renvoie 0.

**4.** En déduire une fonction `decomposition_facteurs_premiers(n)` qui calcule la décomposition en facteurs premiers d'un entier $n \geqslant 2$.
Cette fonction doit renvoyer la liste des couples $(p, v_p(n))$ pour tous les nombres premiers $p$ qui divisent $n$.
Par exemple `decomposition_facteurs_premiers(40)` renvoie la liste `[[2, 3], [5, 1]]`.

# EXERCICE II : Recherche de nombres

## PARTIE A. Recherche de zéro d'une fonction

**1.** Soient $a$ et $b$ deux réels, $f : [a, b] \to \mathbb{R}$ une fonction continue telle que $f(a)f(b) < 0$.

**1.a)** Justifier que $f$ s'annule sur $[a, b]$.

**1.b)** Écrire une fonction Python `rech_dicho` prenant en arguments une fonction `f`, deux flottants `a` et `b` tels que `f(a)f(b)<0` et une précision `eps` et qui renvoie un couple de réels encadrant un zéro de `f` à une précision `eps` près.

**2.** Soit $f$ une fonction continue de $[0, 1]$ dans $[0, 1]$.

**2.a)** Montrer que $f$ admet un point fixe (c'est-à-dire un réel $c$ de $[0, 1]$ tel que $f(c) = c$).

**2.b)** Écrire une fonction Python `rech_pt_fixe` qui prend en argument une fonction `f` que l'on suppose continue de $[0, 1]$ dans $[0, 1]$, un précision `eps` et qui renvoie un couple de réels encadrant un point fixe de `f` à une précision `eps` près. On pourra utiliser la fonction `rech_dicho`.

## PARTIE B. Recherche dans une liste

On propose l'algorithme suivant

```
1  def rech_dicho(L,g,d,x):
2    """L est une liste telle que
         L[g:d+1] est triee"""
3    if x>L[d] :
4      return d+1
5    else :
6      a=g
7      b = d
8      while a != b :
9        c = (a+b)//2
10       if x <= L[c] :
11         b = c
12       else :
13         a = c+1
14   return a
```

**1.** On prend `L = [2,4,5,7,7,9,10]`. Que renvoient les instructions suivantes ?
`>>> rech_dicho(L,1,5,6)`
`>>> rech_dicho(L,0,5,1)`
On donnera les valeurs prises par les variables `a` et `b` à chaque passage ligne 9.

**2.** Détailler clairement ce que fait le programme `rech_dicho`.

**3.** Déterminer, en le justifiant, la complexité du programme, mesurée en nombre de comparaisons. On utilisera, si besoin est, la notation $O$, et on pourra exprimer cette complexité en fonction d'un ou plusieurs paramètres parmi `len(L), g, d, x`.

# EXERCICE III : Nombres premiers jumeaux

**1.** On donne les programmes python P0 et P1 suivants.

```
def P0(N) : # N entier naturel
    if N == 1 :
        return False
    if N == 2 :
        return True
    for d in range(2, N) :
        if N % d == 0 :
            return False
        return True
```

```
def P1(N) : # N entier naturel
    if N == 1 :
        return False
    if N == 2 :
        return True
    for d in range(2, N) :
        if N % d == 0 :
            return False
    return True
```

Que renvoient les appels `P0(5)`, `P1(5)` , `P0(9)` et `P1(9)` ?
Dire en une phrase ce que fait chacun des programmes `P0` et `P1`.

**2.** En une phrase, dire ce que fait le programme python `P2`, qui utilise le programme `P1` précédent :

```
def P2(N) : # N entier naturel
    L = []
    k = 0
    n = k * k + 1
    while n <= N :
        if P1(n) :
            L.append(n)
        k = k + 1
        n = k * k + 1
    return L
```

Que renvoie l'appel `P2(127)` ?

**3.** Écrire une fonction `nextPrime` en langage python qui prend un argument entier $N$ et qui retourne comme valeur le premier nombre premier qui est strictement supérieur à $N$.

**4. Nombres jumeaux**
On appelle *couple de nombres premiers jumeaux* toute liste `[p, q]` telle que `p, q` sont des nombres premiers vérifiant `p < q` et `q = p + 2`. Par exemple `3, 5]`, ou `11, 13]` sont des couples de nombres premiers jumeaux alors que `[2, 3]` ne l'est pas.

**4.a)** Écrire à l'aide de la fonction `nextPrime` précédente, un fonction python nommée `jumeau`, prenant comme argument un entier `N` et renvoyant le couple `[p, q]` de nombres premiers jumeaux tel que `p` strictement supérieur à `N` et le plus petit possible.
Par exemple `jumeau(5)` renvoie comme valeur : `[11, 13]`

**4.b)** Écrire avec les mêmes consignes une fonction, `lesJumeaux`, prenant comme argument un entier `N` et renvoyant la liste de tous les couples de nombres premiers jumeaux `[p, q]` tels que `q` soit inférieur ou égal à `N`.
Par exemple `lesJumeaux(18)`, retourne : `[[3, 5], [5, 7], [11, 13]]`
(le couple `[17, 19]` n'en fait pas partie.)

# EXERCICE I : Valuation $p-$adique

CCP2019 MP2, Exercice 1

```python
def estPremier(n):
    if n<=2:
        return (n==2)
    if n%2 == 0:
        return False
    k=3
    while k**2 <= n:
        if n%k == 0:
            return False
        k += 2
    return True
```

```python
def liste_premier(n):
    L = []
    for k in range(n+1):
        if estPremier(k):
            L.append(k)
    return L
```

```python
def val(n,p):
    nbre = n
    res = 0
    while nbre%p==0:
        res += 1
        nbre = nbre//p
    return(res)
```

```python
def decomposition_facteurs_premiers(n):
    Lp = liste_premier(n)
    Lres = []
    for p in Lp:
        if n%p == 0 :
            Lres.append([p,val(n,p)])
    return Lres
```

# EXERCICE II : Recherche de nombres

E3A 2017 PSI 1, exercice 4

## PARTIE A. Recherche de zéro d'une fonction

**1.** **1.** La fonction $f$ est continue et change de signe sur l'intervalle $[a,b]$ donc, d'après le théorème des valeurs intermédiaires, $\boxed{f \text{ s'annule sur } [a,b]}$.

**2.** La recherche par dichotomie d'un zéro d'une fonction revient à construire les suites $(a_n)$ et $(b_n)$ suivantes, définies par $a_0 = a$, $b_0 = b$ et pour tout $n$ dans $\mathbb{N}$,

$$a_{n+1} = \begin{cases} \dfrac{a_n+b_n}{2} \text{ si } f(a_n)f\left(\dfrac{a_n+b_n}{2}\right) \leqslant 0, \\ a_n \text{ sinon} \end{cases}, \quad b_{n+1} = \begin{cases} b_n \text{ si } f(a_n)f\left(\dfrac{a_n+b_n}{2}\right) \leqslant 0, \\ \dfrac{a_n+b_n}{2} \text{ sinon} \end{cases}$$

D'où le programme suivant

```python
def rech_dicho(f,a,b,eps):
    """f est une fonction continue telle que f(a)*f(b)<0,
    avec a<b, eps un flottant >0"""
    an = a #Initialisation
    bn = b
    while bn-an>eps: # Critere d'arret
        cn=(an+bn)/2
        if f(an)*f(cn)<=0:
            bn = cn
        else:
            an=cn
    return an,bn
```

**2.** **1.** Posons $g : x \mapsto f(x) - x$. Alors $g(0) = f(0) - 0 \geqslant 0$, $g(1) = f(1) - 1 \leqslant 1$. $0 \in [g(1), g(0)]$, $g$ est continue sur l'intervalle $[0,1]$ donc, d'après le théorème des valeurs intermédiaires, on dispose de $\boxed{c \in [0,1] \text{ tel que } g(c) = 0, \text{ i.e. } f(c) = c}$.

**2.** On va utiliser l'idée de la preuve de la question précédente, et appliquer la dichotomie à la fonction $x \mapsto f(x) - x$.

```
def rech_pt_fixe(f,x,eps):
    """f est une fonction de [0,1] dans [0,1], continue"""
    def g(x): #Definition de la fonction auxilliaire
        return f(x) - x
    return rech_dicho(g,0,1,eps)
```

## PARTIE B. Recherche dans une liste

**1.**

```
>>> L = [2,4,5,7,7,8,10]
>>> rech_dicho(L,1,5,6)
a= 1 ,  b= 5
a= 1 ,  b= 3
3
```

```
>>> rech_dicho(L,0,5,1)
a= 0 ,  b= 5
a= 0 ,  b= 2
a= 0 ,  b= 1
0
```

**2.** Le programme `rech_dicho(L,g,d,x)` **recherche** et **renvoie** une **position** `i` entre `g` et `d` telle que `L[i-1]`$\leqslant$ `x` $\leqslant$ `L[i]`, sachant que l'on suppose `L[g:d+1]` triée.

**3.** Si $a_k$ et $b_k$ sont les valeurs de `a` et `b` à la $k$-ième étape, alors $\dfrac{a_k + b_k}{2} - 1 \leqslant c_k \leqslant \dfrac{a_k + b_k}{2}$. Alors selon les deux cas de figure du test `if`, on obtient

$$b_{k+1} - a_{k+1} = c_k - a_k \leqslant \frac{a_k + b_k}{2} - a_k \leqslant \frac{b_k - a_k}{2},$$
$$\text{ou bien } b_{k+1} - a_{k+1} = b_k - (c_k + 1) = b_k - c_k - 1 \leqslant b_k - \frac{a_k + b_k}{2} \leqslant \frac{b_k - a_k}{2},$$

donc, par une récurrence immédiate, $b_k - a_k \leqslant 2^{-k}(d - g + 1)$, donc le programme s'arrÃªte lorsque $b_k - a_k < 1$, en particulier lorsque $2^{-k}(d - g + 1) < 1$, i.e. $k \geqslant \log_2(d - g + 1)$, d'où une complexité en $O(\ln(d - g))$.

# EXERCICE III : Nombres premiers jumeaux

E3A PSI 2016, ex3

## 1. Programmes mystères

**1.** On a `P0(5)=True` et `P1(5)=True` puis `P0(9)=True` et `P1(9)=False`
`P0(N)` pour `N` $\geqslant 3$ teste si `N` est pair. Il renvoie `False` si c'est le cas et `True` sinon.
`P1(N)` teste si `N` est premier (en renvoyant `True` si c'est le cas).

**2.** `P2(N)` renvoie la liste de entiers premiers inférieurs à `N` qui sont de la forme $1 + k^2$ pour une un bon entier $k$. On a ainsi

`P2(127)=[2,5,17,37,101]`

*On teste qui parmi* $1 + 0^2, 1 + 1^2, 1 + 2^2, 1 + 3^2, \ldots, 1 + 11^2$ *est premier (au delà, on est après* 127*).*

**3.** On gère un compteur que l'on incrémente à partir de la valeur $N+1$ tant que l'on ne tombe pas sur un nombre premier.

```
def nextPrime(N):
    k=N+1
    while not P1(k):
        k=k+1
    return k
```

**4. 4.a)** Je gère deux variables `p` et `q` qui vont successivement prendre les valeurs des entiers premiers successifs (plus grands que `N`). On s'arrête quand il sont distants de 2.

```
def jumeau(N):
    p=nextPrime(N)
    q=nextPrime(p)
    while q!=p+2:
        p,q=q,nextPrime(q)
    return([p,q])
```

On peut aussi tester pour les nombre premiers $\texttt{p} > \texttt{N}$ si $\texttt{p}+2$ est premier. Cela fait utiliser `P1`.

```
def jumeau(N):
    p=nextPrime(N)
    while not(P1(p+2)):
        p=nextPrime(p)
    return([p,p+2])
```

**4.b)** Je gère deux variables `p` et `q` qui vont successivement prendre les valeurs des entiers premiers successifs et une liste `l` pour stocker les paires de jumeaux. Tant que $\texttt{q} \leqslant \texttt{N}$, on teste si $\texttt{p}+2 = \texttt{q}$ pour savoir si on l'ajoute à la liste puis on passe au couple suivant.

```
def lesJumeaux(N):
    p=2
    q=nextPrime(p)
    l=[]
    while (q<=N):
        if q==p+2:l.append([p,q])
        p,q=q,nextPrime(q)
    return(l)
```