Les fonctions premiers pas .

Exercice 1

Écrire deux fonctions qui convertissent en degrés Celsius une température exprimée au départ en degrés Farenheit et l'inverse. La formule de conversion est $T_F = 1,8T_C + 32$

```
def CeltoFar(T):
    return 1.8 * T + 32
```

```
def FartoCel(T):
    return (T - 32)/1.8
```

Exercice 2

1. Écrire une fonction puissance prenant en entier n et un réel x et renvoyant x^n.

```
def puiss(x,n):
    p= 1
    for i in range(0,n):
        p=p* x
    retur n p
```

Combien y a t-il d'affectations ? Il y a n+1 affectations

2. Écrire une fonction prenant en entrée un entier n et renvoyant $n!$.

```
def facto(n):
    p=1
    for i in range(1,n+1):
        p=p*i
    return p
```

Combien y a t-il d'affectations ? Il y a $n+1$ affectations.

3. Écrire une fonction prenant en entrée deux entiers n et p et renvoyant $\binom{n}{p}$ pour $p \leq n$.
On utilisera la fonction précédente.

```
def binom(n,p):
    if p<=n:
        return facto(n)/(facto(p)*facto(n-p))
```

⚠ Il y a $(n+1)+(p+1)+(n-p+1) = 2n+3$ affectations !! Il faut compter celle de chaque facto !!

4. Écrire une fonction prenant en entrée un entier n et un réel x et renvoyant $S_n = \sum_{k=0}^{n} \frac{x^k}{k!}$.
On fera deux programmes (un utilisant les fonctions précédentes et un ne les utilisant pas).

```
def somme1(n,x):
    s=0
    for k in range(0,n+1):
        s= s + puiss(x,k)/facto(k)
    return s
```

⚠ Cette fonction n'est pas efficace... facto(k) recalculé à chaque fois k!

Il y a donc $\sum_{k=0}^{n} \underbrace{\left(\underbrace{(k+1)}_{\text{puiss}(x,k)} + \overbrace{(k+1)}^{\text{facto}(k)}\right)}_{\text{chaque passage dans la boucle for}} = 2\sum_{k=0}^{n}(k+1)$

$= 2\frac{(n+1)(n+2)}{2} = 6(n^2)$ affectations !

```
def somm2(n,x):
    s= 1
    p= 1
    f= 1
    for k in range(1,n+1):
        p= p*x
        f= f*i
        s = s + p/f

    return s
```

Il y a donc $n \times 3 = 3n$ affectations
$O(n)$.

Préciser le nombre d'affectations dans chaque cas.

Exercice 4

1. Écrire une fonction qui détermine si un nombre est premier .

```
def premier(n):
    d=2
    while d<=sqrt(n):
        if n%d==0:                  # le reste de la division euclidienne est nul
            return False
        d=d+1
    return True                     # True et False sont des booleens.
```

Respecter la taille.

2. Écrire une fonction qui prend en entrée une liste de nombres entiers et qui supprime les éléments premiers de cette liste.

```
def supprem(L):
    i=0
    while i<=len(L)-1 :             # ⚠ len(L)-1 avec les décalages !
        if premier(L[i]):
            del(L[i])
        i=i+1
    return L
```

⚠ Il faut obligatoirement un WHILE car la longueur len(L) varie à chaque passage puisque des éléments sont supprimés.

Exercice 5

Le célèbre crible d' Eratosthène pour trouver tous les nombres premiers inférieurs a un entier n consiste à écrire tous les entiers inférieurs ou égaux à n, puis rayer les multiples de 2, ceux de 3 et ensuite de chaque nombre non rayé qui est premier. A la fin il ne reste que les nombres premiers. Programmez le crible d' Eratosthène pour le nombres premiers à partir d'une liste en remplaçant au fur et à mesure tous les nombres non premiers par des *. Affichez le résultat sous la forme d'un tableau grâce à la fonction précédente.

```
L = [1,2,3,4,5,6,7,8, . . . . . ]

def crible(L):
    for i in range(2, len(L)):
        for j in range(i+1, len(L)):      # depart à i+1
            if L[j] != '*':               # important pour pouvoir tester la divisibilité !
                if L[j] % i == 0:
                    L[j] = "*"

    return L


def prem(N):
    L = range(1, N+1)
    L = crible(L)
    P = []
    for x in L:
        if x != "*":
            P.append(x)

    return P
```