```
# ============================== #
# PSI*                           #
# DS d'informatique du 18/11/2017 #
# Corrigé de l'exercice 2        #
# ============================== #


# ---------------- #
# --> PARTIE I <-- #
# ---------------- #

## Question 2

def binomial_base(n,p):
    # Test sur le type des arguments (non demandé dans le DS)
    if type(n) != int or type(p) != int:
        raise TypeError("ATTENTION ! Les deux arguments doivent être des
entiers !")
    # Test sur la positivité des arguments (non demandé dans le DS)
    elif n < 0 or p < 0:
        raise ValueError("ATTENTION ! Les deux arguments doivent être des
entiers naturels !")
    # Comparaison de p et n
    elif p > n:
        raise ValueError("Le deuxième argument doit être inférieur au
premier !")
    # Les deux arguments sont bien des entiers naturels et p est inférieur à
n.
    else:
        if p > n/2:
            p = n - p
        num, denum = 1, 1
        for k in range(p):
            num *= (n - k)
            denum *= (k + 1)
        return(num//denum)

# ----------------- #
# --> PARTIE II <-- #
# ----------------- #

## Question 3

def binomial_rec1(n,p):
    # Test sur le type des arguments (non demandé dans le DS)
    if type(n) != int or type(p) != int:
        raise TypeError("ATTENTION ! Les deux arguments doivent être des
entiers !")
    # Test sur la positivité des arguments (non demandé dans le DS)
    elif n < 0 or p < 0:
        raise ValueError("ATTENTION ! Les deux arguments doivent être des
entiers naturels !")
    # Comparaison de p et n
    elif p > n:
        raise ValueError("Le deuxième argument doit être inférieur au
premier !")
    # Les deux arguments sont bien des entiers naturels et p est inférieur à
n.
    else:
        if p == n or p == 0:
```

```
            return(1)
        else:
            return(binomial_rec1(n-1,p) + binomial_rec1(n-1,p-1))

## Question 5

def binomial_rec2(n,p):
    # Test sur le type des arguments (non demandé dans le DS)
    if type(n) != int or type(p) != int:
        raise TypeError("ATTENTION ! Les deux arguments doivent être des
entiers !")
    # Test sur la positivité des arguments (non demandé dans le DS)
    elif n < 0 or p < 0:
        raise ValueError("ATTENTION ! Les deux arguments doivent être des
entiers naturels !")
    # Comparaison de p et n (non demandé dans le DS)
    elif p > n:
        raise ValueError("Le deuxième argument doit être inférieur au
premier !")
    # Les deux arguments sont bien des entiers naturels et p est inférieur à
n.
    else:
        if p == 0 or n == p:
            return 1
        else:
            # ATTENTION ! Il faut écrire très soigneusement l'appel récursif
!
            # On multiplie D'ABORD n par le résultat de l'appel récursif
afin
            # d'être certain que le résultat sera bien divisible par p et on
            # effectue ensuite une division entière afin d'obtenir...
l'entier
            # souhaité !
            return((n * binomial_rec2(n-1,p-1)) // p)

# ------------------ #
# --> PARTIE III <-- #
# ------------------ #

## Question 6

def Gamma_LN(x):
    # x doit être numérique et strictement positif (tests non demandés dans
le DS)
    if type(x) != int and type(x) != float:
        raise TypeError("L'argument doit être numérique !")
    elif x <= 0:
        raise ValueError("L'argument doit être strictement positif !")
    # L'argument est bien un numérique strictement positif
    else:
        from math import pi, log
        # La liste des coefficients de LANCZOS
        L = [1.000000000190015,
            76.18009172947146,
           -86.50532032941677,
            24.01409824083091,
            -1.231739572450155,
             0.001208650973866179,
            -0.000005395239384953]
        y = x + 5.5
```

```
        res = -y + (x + 0.5) * log(y) - log(x) + 0.5 * log(2 * pi)
        y2 = L[0]
        for i in range(1,7):
            y2 += L[i] / (x + i)
        res += log(y2)
        return(res)

## Question 7

def factorial_wg(n):
    # n doit être un entier positif (tests non demandés dans le DS)
    if type(n) != int:
        raise TypeError("L'argument doit être un entier !")
    elif n < 0:
        raise ValueError("L'argument doit être un entier POSITIF !")
    # L'argument est bien un entier positif
    else:
        from math import exp
        F = [1.,1.,2.,6.,24.]
        if n <= 4:
            return(F[n])
        elif n <= 30:
            res = 24.
            for k in range(5,n+1):
                res *= k
            return(res)
        else:
            return(exp(Gamma_LN(n+1)))

## Question 8

def factorial_LN_wg(n):
    # n doit être un entier positif (tests non demandés dans le DS)
    if type(n) != int:
        raise TypeError("L'argument doit être un entier !")
    elif n < 0:
        raise ValueError("L'argument doit être un entier POSITIF !")
    # L'argument est bien un entier positif
    else:
        from math import log
        F = [1.,1.,2.,6.,24.]
        if n <= 4:
            return(log(F[n]))
        elif n <= 30:
            res = 24.
            for k in range(5,n+1):
                res *= k
            return(log(res))
        else:
            return(Gamma_LN(n+1))

## Question 9

def binomial_wg(n,p):
    # Test sur le type des arguments (non demandé dans le DS)
    if type(n) != int or type(p) != int:
        raise TypeError("ATTENTION ! Les deux arguments doivent être des
entiers !")
    # Test sur la positivité des arguments (non demandé dans le DS)
    elif n < 0 or p < 0:
```

```
        raise ValueError("ATTENTION ! Les deux arguments doivent être des
entiers naturels !")
    # Comparaison de p et n (non demandé dans le DS)
    elif p > n:
        raise ValueError("Le deuxième argument doit être inférieur au
premier !")
    # Les deux arguments sont bien des entiers naturels et p est inférieur à
n.
    else:
        from math import floor, exp
        return(float(floor(0.5 + exp(factorial_LN_wg(n)-factorial_LN_wg(p)-
factorial_LN_wg(n-p)))))
```