```
# ============================================== #
# ============================================== #
# IPT MATHS SPE                                  #
# Introduction (Séance I de 2017-2018)           #
# Exercices de niveau II - Propositions de code  #
# Septembre 2017                                 #
# ============================================== #
# ============================================== #

# ============= #
# EXERCICE II-1 #
# ============= #

def IsArithSeq(L):
    l = len(L)
    r = L[1] - L[0]
    i = 1
    while i <= l-2 and L[i+1] - L[i] == r:
        i += 1
    return(i == l - 1)

def IsGeomSeq(L):
    l = len(L)
    q = L[1] / L[0]
    i = 1
    while i <= l-2 and L[i+1] / L[i] == q:
        i += 1
    return(i == l - 1)

# ============= #
# EXERCICE II-2 #
# ============= #

# Avec une liste passée en argument

def IsPalindrome(L):
    while L != [] and L[0] == L[-1]:
        del L[0]
        if L != []:
            del L[-1]
    return(L == [])

def IsPalindrome_rec(L):
    if L == [] or len(L) == 1:
        return(True)
    else:
        if L[0] == L[-1]:
            del L[0]
            if len(L) != 1:
                del L[-1]
            return IsPalindrome_rec(L)
        else:
            return(False)

# ============= #
# EXERCICE II-3 #
# ============= #

def StringDecompo(cc):
    V = 'aeiouy'
```

```
    vo, co = '', ''
    for e in cc:
        if e in V:
            vo += e
        else:
            co += e
    return(vo,co)

# ============= #
# EXERCICE II-4 #
# ============= #

def Euler(epsilon):
    from math import log1p
    N = 1
    d = 0.5 - log(2)
    while abs(d) >= epsilon :
        N += 1
        d = 1/(N+1) - log1p(1/N)
    return N

# ============= #
# EXERCICE II-5 #
# ============= #

def Heron(a,b,epsilon):
    # Pour éviter que la boucle principale ne soit exécutée un trop grand
nombre
    # de fois, on limite la valeur maximale de N à 1000 (largement suffisant
!).
    N = 0
    # valeur absolue de la différence initiale : |u(1) - u(0)|
    u = b
    new_u = (b + a / b) * 0.5
    d = abs(new_u - u)
    # boucle principale
    while d >= epsilon and N < 1000:
        N += 1
        u = new_u
        new_u = (u + a / u) * 0.5
        d = abs(new_u - u)
    # La fonction renvoie un tuple
    return(N,u)

# ============= #
# EXERCICE II-6 #
# ============= #

def EcartMax(L):
    em = 0
    imax, jmax = 0, 0
    for i in range(len(L)-1):
        for j in range(i+1,len(L)):
            d_curr = abs(L[i] - L[j])
            if d_curr > em:
                imax, jmax, em = i, j, d_curr
    return(em, imax, jmax)

# ============= #
# EXERCICE II-7 #
```

```
# ============= #

def nPlotRac(n):
    # Importation des éléments requis depuis le module math
    from math import pi, cos, sin
    # Importation du module pyplot de la bibliothèque matplotlib
    import matplotlib.pyplot as plt
    # Construction
    X, Y = [], []
    for k in range(n+1):
        teta = 2*k*pi/n
        X.append(cos(teta))
        Y.append(sin(teta))
    # Nettoyage de la fenêtre graphique
    plt.clf()
    # construction des divers éléments graphiques
    plt.axes(axisbg="black")
    plt.grid(color="grey", linewidth=2, linestyle='-')
    plt.title("Polygone régulier dont les sommets sont les points
d'affixes\nles racines "+str(n)+"ièmes de l'unité")
    # Affichage du polygone et des divers éléments graphiques
    plt.plot(X,Y,linewidth=2,color="red")
    plt.show()

# ============= #
# EXERCICE II-8 #
# ============= #

# RAPPEL : dans la fonction ConvexValid, nous faisons l'hypothèse que 3
sommets
# consécutifs ne peuvent être alignés.

## Opérateur DELTA

def DELTA(S1,S2,P):
    # RAPPEL : S1, S2 et P sont des tuples comportant chacun deux éléments
    # Cas général : les sommets S1 et S2 n'ont pas la même abscisse
    if S2[0] != S1[0]:
        r = (S2[1]*(P[0]-S1[0])+S1[1]*(S2[0]-P[0]))/(S2[0]-S1[0])-P[1]
    # Cas particulier : les sommets S1 et S2 ont la même abscisse
    else:
        r = P[0] - S1[0]
    # return approprié
    if r > 0:
        return(1)
    else:
        return(-1)

## Fonction ConvexValid (avec représentation graphique par défaut)

def ConvexValid(S,g=True):
    # Nombre de sommets du polygone
    N = len(S)
    if N <= 3:
        return("ATTENTION ! Votre polygone doit comporter au moins 4 sommets
!")
    else:
        for i in range(N-1):
            n_plus, n_moins = 0, 0
            # Sommets d'indices 0 à i-1
```

```
            for j in range(i):
                if DELTA(S[i],S[i+1],S[j]) == 1:
                    n_plus += 1
                else:
                    n_moins += 1
                if n_plus * n_moins != 0:
                    return(False)
            # Sommets d'indices i+2 à N-1
            for j in range(i+2,N):
                if DELTA(S[i],S[i+1],S[j]) == 1:
                    n_plus += 1
                else:
                    n_moins += 1
                if n_plus * n_moins != 0:
                    return(False)
        # Cas du dernier sommet
        n_plus, n_moins = 0, 0
        for j in range(1,N-1):
            if DELTA(S[N-1],S[0],S[j]) == 1:
                n_plus += 1
            else:
                n_moins += 1
            if n_plus * n_moins != 0:
                return(False)
        # Cette ligne étant atteinte, le polygon est convexe !
        if g:
            # Si le 2ème argument a été positionné à True (valeur par
défaut),
            # on affiche le polygone.
            from matplotlib import pyplot as plt
            # on peut aussi utiliser  la syntaxe :
            # import matplotlib.pyplot as plt
            plt.clf()
            X, Y = [], []
            for i in range(N):
                X.append(S[i][0])
                Y.append(S[i][1])
            X.append(S[0][0])
            Y.append(S[0][1])
            plt.plot(X,Y,"r")
            plt.show()
        return(True)

# ============= #
# EXERCICE II-9 #
# ============= #

def syracuse(u):
    print(u)
    while u != 1:
        if u%2 == 0:
            u = u // 2
        else:
            u = 3 * u + 1
        print(u)

def syracuse2(u):
    u0, n = u, 0
    print(u)
    T_vol, T_volAlt, max = 0, 0, u
```

```
while u != 1:
    n += 1
    if u%2 == 0:
        u = u // 2
    else:
        u = 3 * u + 1
    print(u)
    if u > max:
        max = u
    if u <= u0 and T_volAlt == 0:
        T_volAlt = n - 1
    T_vol += 1
print('Le cycle limite est atteint !')
# Plus petit indice n tel que u(n) = 1
print('Temps de vol :',T_vol)
# Plus petit indice n tel que u(n+1) <= u(0)
print('Temps de vol en altitude :',T_volAlt)
# Valeur maximale de la suite
print('Altitude maximale :',max)
```