```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  5 09:35:03 2022

@author: stephane

Python 3.8.10 (default, Jun 22 2022, 20:18:18)
"""

from typing import List
import matplotlib.pyplot as pypl  # Exo 5
from numpy import arange          # Exo 5
from math import sin              # Exo 5
from numpy.random import randint  # Exo 14

# Exo 1 : DONE

# Exo 2

somme1 = 0
for k in range(101):
    somme1 = somme1 + k**2

somme2 = sum(k**2 for k in range(101))

"""
>>> somme1
338350

>>> somme2
338350
"""


# Exo 3

somme3 = 0
for k in range(100, 1001):
    if (k**2 + 2*k + 1) % 42 == 0:
        somme3 += k

somme4 = sum([k for k in range(100, 1001) if (k**2 + 2*k + 1) % 42 == 0])

"""
>>> somme3, somme4
(11445, 11445)
"""

# Exo 4

def f4(x: float) -> float:
    return x**2+2

"""
>>> f4(100)
10002
"""

# Exo 5

def f(x: float) -> float:
```



```
    if x <= 0:
        return 20*sin(x/5)
    elif x <= 10:
        return x
    elif x <= 20:
        return 5*x-40
    else:  # optionnel : pourquoi ?
        return -3*x+120

les_x = arange(-50, 50, 0.1)
les_y = [f(x) for x in les_x]
pypl.plot(les_x, les_y)

pypl.grid()
pypl.axhline(color='black')
pypl.axvline(color='black')
pypl.savefig('graphe_f.pdf')

# Exo 6

def facto1(n: int) -> int:
    if n == 0:  # Ne pas oublier le cas de base/final
        return 1
    return n*facto1(n-1)

def facto2(n: int) -> int:
    produit = 1
    for k in range(1, 1+n):
        produit *= k
    return produit

"""
>>> facto1(10), facto2(10)
(3628800, 3628800)
"""



# Exo 7

t1 = list(range(1, 30, 2))
t2 = [x**2 for x in t1]
t3 = [y for y in t2 if y % 12 == 1]
t4 = [[2**i*(2*j+1) for j in range(4)] for i in range(5)]

"""
>>> t1
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]

>>> t2
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361, 441, 529, 625, 729, 841]

>>> t3
[1, 25, 49, 121, 169, 289, 361, 529, 625, 841]

>>> t4
[[1, 3, 5, 7],
 [2, 6, 10, 14],
 [4, 12, 20, 28],
 [8, 24, 40, 56],
 [16, 48, 80, 112]]
"""
```

```
# Exo 8

def somme(t: List) -> float:
    s = 0
    for x in t:
        s = s + x
    return s

def produit(t: List) -> float:
    p = 1
    for x in t:
        p = p * x
    return p

def maximum(t: List) -> float:
    assert t != []  # signifie : «ne va pas plus loin si la liste est vide»
    maxi = t[0]
    for x in t:
        if x > maxi:
            maxi = x
    return maxi

def indice_maximum(t: List) -> int:
    assert t != []
    i_maxi = 0
    for i in range(1, len(t)):
        if t[i] > t[i_maxi]:
            i_maxi = i
    return i_maxi

"""
>>> somme(t1), produit(t1), maximum([10, 42, 5, 42, 30]),
    indice_maximum([10, 42, 5, 42, 30])
(225, 6190283353629375, 42, 1)
"""


# Exo 9

def inversions(t: List) -> int:  # Calcul du nombre d'inversions
    cpt = 0
    for j in range(1, len(t)):
        for i in range(j):
            if t[j] < t[i]:
                cpt += 1
    return cpt

"""
>>> inversions([1, 2, 3, 0]), inversions([4, 3, 2, 1])
(3, 6)
"""

# Exo 10

t10 = [1]
for j in range(1, 11):
    t10.append(sum((i+1)*t10[j-1-i] for i in range(j)))

"""
>>> t10
[1, 1, 3, 8, 21, 55, 144, 377, 987, 2584, 6765]
```



```
"""

# Exo 11

def pascal(n: int) -> List[List[int]]:
    if n == 0:
        return [[1]]
    avant = pascal(n-1)   # les «k parmi i» pour 0 <= k <= i <= n-1
    derniere = avant[-1]  # dernière ligne : les «k parmi n-1»
    nouvelle = [1]        # bord gauche
    for k in range(1, n):
        nouvelle.append(derniere[k-1] + derniere[k])  # formule du binôme
    nouvelle.append(1)   # bord droit
    avant.append(nouvelle)
    return avant

"""
>>> pascal(4)
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
"""

# Exo 12

def composante(s0: int, G: List[List[int]]) -> List[int]:
    comp = [s0]
    i = 0  # l'indice du sommet à traiter
    while i <= len(comp)-1:
        s = comp[i]
        for j in G[s]:
            if not j in comp:
                comp.append(j)
        i = i+1
    return comp

G0 = [[5], [2, 4], [1], [], [1], [0]]

"""
>>> composante(0, G0)
[0, 5]

>>> [composante(s, G0) for s in range(6)]
[[0, 5], [1, 2, 4], [2, 1, 4], [3], [4, 1, 2], [5, 0]]
"""

# Exo 13

Graphe = List[List[int]]

def composantes(G: Graphe) -> List[List[int]]:
    # On tient à jours les composantes, les sommets vus, et le prochain à voir
    comps, vus, prochain = [], [], 0
    while prochain < len(G):
        if not(prochain in vus):
            cp = composante(prochain, G)
            comps.append(cp)
            vus.extend(cp)
        prochain += 1
    return comps

"""
>>> composantes(G0)
[[0, 5], [1, 2, 4], [3]]
```

```
"""

# Exo 14

def creer_graphe(nb_sommets: int, nb_aretes: int) -> Graphe:
    G = [[] for _ in range(nb_sommets)]
    for _ in range(nb_aretes):
        i, j = randint(nb_sommets), randint(nb_sommets)
        if i != j and not(j in G[i]):
            G[i].append(j)
            G[j].append(i)
    return(G)

"""
>>> G1 = creer_graphe(10, 10)

>>> G1
[[6], [7, 6], [6], [5], [7], [3], [1, 2, 0], [8, 1, 4], [7], []]

>>> composantes(G1)
[[0, 6, 1, 2, 7, 8, 4], [3, 5], [9]]

"""

def ecrire_graphe(G: Graphe, nom_fichier: str):
    fout = open(nom_fichier, 'w')
    for ligne in G:
        fout.write(str(ligne)[1:-1] + '\n')
    fout.close()

def lire_graphe(nom_fichier:str) -> Graphe:
    gr = []
    for ligne in open(nom_fichier):
        casse = ligne.strip().split(',')
        if casse[0] == '':
            gr.append([])
        else:
            gr.append([int(x) for x in casse])
    return gr

"""
>>> ecrire_graphe(G1, 'G10_10.gr')

>>> lire_graphe('G10_10.gr')
[[6], [7, 6], [6], [5], [7], [3], [1, 2, 0], [8, 1, 4], [7], []]

>>> ecrire_graphe(creer_graphe(1000, 2000), 'G2000_1000.gr')
>>> grosG = lire_graphe('G2000_1000.gr')
>>> len(composante(0, grosG))
977

>>> [len(c) for c in composantes(grosG)]
[977, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

>>> len(composantes(grosG))
24

>>> [len(c) for c in composantes(creer_graphe(1000, 10000))]
[1000]

>>> [len(c) for c in composantes(creer_graphe(1000, 5000))]
[1000]
```



```
>>> [len(c) for c in composantes(creer_graphe(1000, 2000))]
[973, 1, 1, 1, 1, ... 1, 1]

>>> [len(c) for c in composantes(creer_graphe(1000, 1000))]
[1, 788, 2, 1, 3, 6, 14, 1, 11, 1, 1, ..., 1, 1, 1, 1]
"""
```