nphysic_7a.py

```
#!/usr/bin/env python3.6
# -*- coding: utf-8 -*-
# Série communs. En Général Python

import time
import copy

"""MusicAToumic En"""
# Programme       nphysic_4a : 3 juillet 2017
# Programme        nphysic_5 : 16 juillet 2017
# Programme       nphysic_7a : 30 juillet 2017
"""Découvrir les multiples d'un NOMBRE

1- Tableau(imem[NP]) des :ip;dv: de type (1, 5)
2- Copie (imem) sur tableau(itab)
3- Boucle (imem) sur (itab)
4- :ion1::ion2: Motif de progression
5- Modélisation :ip:
6- :print: Points d'analyses

Typogramme nucloïde

P = Points d'itérations 'u15171w-14925...'
NP = Nombre Pemier: Signé '°'
NpP = Nombre préalable Premier: Types '1&5'
NpPip = Tableau :imem[ip(NP)]: Résultat "ip"
NpPdv = Tableau :imem[dv(NP)]: Résultat "ip"
NC = Nombre Commun :itab[NC]: Résultat "ip"

Les premiers tempéraments préalables NpP '1&5'
Fonction :imemap: Nombres Premiers NP
Les multiplications NP produisent NC
Mes premiers pas sur sur cette échelle
"""

dod0 = dod1 = time.time()


def onuno(uno):
    """ Pour un nombre :uno:
    À la recherche des sous-multiples
    """
    u1 = 0
    if uno == 0:  # :uno=0: Zéro issue
        uno = 1
        u1 = 1

    uo5 = so5 = int(uno ** .5)
    uo6 = uno % 6
    u25 = s25 = int(uno ** .25)
    iut = sut = ((uo5 - u25) // 2)
    # print('uo5(v):', uo5, 'u25(w,x):', u25, 'iut(y,z):', iut)

    """
    La partie suivante fabrique trois groupes liés (sut,so5,s25)
    Chaque groupe a un "[0]Type(1&5)". Sert au tri (1&5).
    "[1]Type(1)|[2]Type(5)". À raison des compteurs incrémentés :6:
    Et, en incrémentant de six chaque lignée (1&5), on ne rate pas
```

```
le nombre préalable se trouvant sur la lignée(1), (occasionnel
lorsqu'on incrémente uniquement la lignée (5)).
"""

# Initialisations des tableaux-types
s00 = []  # Goupe :sut:
s01 = []  # Goupe :so5:
s02 = []  # Goupe :s25:
if int(sut % 6) in (1, 5):
    s00.append(int(sut % 6))
    if int(sut % 6) == 1:
        s00.append(sut)
        s00.append(sut + 4)
    else:
        s00.append(sut + 2)
        s00.append(sut)
else:
    while 1:
        sut += 1
        if int(sut % 6) in (1, 5):
            s00.append(int(sut % 6))
            s00[0] = int(sut % 6)
            # :s00[0]: = nombre type état
            if int(sut % 6) == 1:
                s00.append(sut)
                s00.append(sut + 4)
                # :s00[0]: = type 1
                # :s00[1]: = nombre type 1
                # :s00[2]: = nombre type 5
            else:
                s00.append(sut + 2)
                s00.append(sut)
                # :s00[0]: = type 5
                # :s00[1]: = nombre type 1
                # :s00[2]: = nombre type 5
            break

if int(so5 % 6) in (1, 5):
    s01.append(int(so5 % 6))
    # :s01[0]: = nombre type état
    if int(so5 % 6) == 1:
        s01.append(so5)
        s01.append(so5 + 4)
        # :s01[0]: = type 1
        # :s01[1]: = nombre type 1
        # :s01[2]: = nombre type 5
    else:
        s01.append(so5 + 2)
        s01.append(so5)
        # :s01[0]: = type 5
        # :s01[1]: = nombre type 1
        # :s01[2]: = nombre type 5
else:
    while 1:
        so5 += 1
        if int(so5 % 6) in (1, 5):
            s01.append(int(so5 % 6))
            if int(so5 % 6) == 1:
                s01.append(so5)
                s01.append(so5 + 4)
            else:
```

```
                s01.append(so5 + 2)
                s01.append(so5)
            break

if int(s25 % 6) in (1, 5):
    s02.append(int(s25 % 6))
    # :s02[0]: = nombre type état s25
    if int(s25 % 6) == 1:
        s02.append(s25)
        s02.append(s25 + 4)
        # :s02[0]: = type 1
        # :s02[1]: = nombre type 1
        # :s02[2]: = nombre type 5
    else:
        s02.append(s25 + 2)
        s02.append(s25)
        # :s02[0]: = type 5
        # :s02[1]: = nombre type 1
else:
    while 1:
        s25 -= 1
        if int(s25 % 6) in (1, 5):
            s02.append(int(s25 % 6))
            if int(s25 % 6) == 1:
                s02.append(s25)
                s02.append(s25 + 4)
            else:
                s02.append(s25 + 2)
                s02.append(s25)
            break

# Affectations des compteurs-types
"""v = s01[1]  # :so5: Racine(uno). Type 1
w = x = s02[1]  # :s25: Racine(so5). Type 1
:v5,y5,w5,z5,x5: Sont de type 5
"""
# Section :v >= y:
v = s01[1]  # Nombre type 1
v5 = s01[2]  # Nombre type 5
y = s00[1]  # Nombre type 1
y5 = s00[2]  # ...
# Section :w >= u:
""":u: est incrémenté de 1"""
w = s02[1]
w5 = s02[2]
# Section :z >= x:
z = s00[1]
z5 = s00[2]
x = s02[1]
x5 = s02[2]
# print('s01:%s s00:%s s02:%s' % (s01, s00, s02))

# Tableau :imem[ip(NP)]:
imem = []
if iut > u25:  # Si oui: Commence à 1
    u = 1
else:  # Si non: Commence à 2. Sinon ERREUR
    u = 2

v_ = w_ = z_ = 0  # Indices des fins (v_ pour v...)
if u1 == 0:  # Zéro issue
```

```
# Fonction d'appel à condition NP
def imemap(uz):
    """ Les nombres premiers NP
    Des préalables premiers :uz: déjà '1&5' """
    # Condition du niveau bas (1, 2, 3, 4, 5, 6, 7)
    if not uno % uz and uz % 6 in (1, 3, 5) \
       and uz < 8:
        imem.append(uz)  # Tableau :imem[ip(NP)]:
    else:
        # print(uz, 'mim2', imem)
        # :rim: Reconstruction à chaque tour
        rim = [mm for mm in imem if mm > 1]
        for rr in rim:
            if not uz % rr:
                r0 = 1
                break
        else:
            # Rapports des racines carrées :uz:
            sq5 = int(uz ** .5)
            sq2 = int(uz ** .25)
            i0 = (sq5 - sq2) // 2  # Entre :sq2: et :sq5:
            # :iep: Ordonnée de carré à carré²
            # print('sq5:%s sq2:%s i0:%s' % (sq5, sq2, i0))
            q = 0
            for iep in range(sq5, i0, -1):
                if not uz % iep \
                   and iep % 6 in (1, 5):
                    r0 = 1
                    break
                if not uz % (uz // iep) \
                   and (uz // iep) % 6 in (1, 5):
                    r0 = 1
                    break
                o = sq2 + q  # Ordonnée de carré² incrémenté
                if o > 1:
                    if not uz % o \
                       and o % 6 in (1, 5):
                        r0 = 1
                        break
                    if not uz % (uz // o) \
                       and (uz // o) % 6 in (1, 5):
                        r0 = 1
                        break
                p = sq2 - q  # Ordonnée de carré² décrémenté
                if p > 1:
                    if not uz % p \
                       and p % 6 in (1, 5):
                        r0 = 1
                        break
                    if not uz % (uz // p) \
                       and (uz // p) % 6 in (1, 5):
                        r0 = 1
                        break
                q += 1
            else:
                r0 = 0
        if r0 == 0:  # :for: Condition vide, donc.
            imem.append(uz)
            # Nombre premier NP

# À la recherche des sous-multiples
```

```
while 1:
    # Ordonnée conditionnée aux sous-multiples
    # Condition du niveau bas (1, 2, 3, 4, 5)
    if not uno % u and u % 6 in (1, 2, 3, 5) \
       and u < 6 and u not in imem:
        imem.append(u)  # Tableau :imem[ip(NP)]:
    # :v >= y::sq5|sut:
    if v >= y and v_ == 0:  # Si oui: Faire
        if not uno % v and v not in imem:
            imemap(v)  # Appel :imemap: Sortant :v: NpP
            if v == uo5:  # Le carré est rencontré
                imemap(v)
        if not uno % v5 and v5 not in imem:
            imemap(v5)  # Appel :imemap: Sortant :v5: NpP
            if v5 == uo5:  # Le carré est rencontré
                imemap(v5)
        if not uno % y \
           and y not in imem:  # y = iut(+1)
            imemap(y)
        if not uno % y5 \
           and y5 not in imem:  # y5 = iut(+1)
            imemap(y5)
    else:  # Si non: Indice terminal
        v_ = 1

    # :w >= u::s25|u:
    if w >= u - 6 and w_ == 0:  # Si oui: Faire
        if not uno % w \
           and w not in imem:  # w = u25(-1)
            imemap(w)  # Appel :imemap: Sortant :w: NpP
        if not uno % u and u not in imem \
           and u % 6 in (1, 5) \
           and u > 1:  # u = u(+1)
            imemap(u)
        if not uno % w5 and w5 > 1 \
           and w5 not in imem:  # w5 = u25(-1)
            imemap(w5)  # Appel :imemap: Sortant :w5: NpP
    else:  # Si non: Indice terminal
        w_ = 1

    # :z >= x::sut|s25:
    if z >= x and z_ == 0:  # Si oui: Faire z_
        if not uno % z \
           and z not in imem:  # z = iut(-1)
            imemap(z)
        if not uno % x \
           and x not in imem:  # x = u25(+1)
            imemap(x)
        if not uno % z5 \
           and z5 not in imem:  # z5 = iut(-1)
            imemap(z5)
        if not uno % x5 \
           and x5 not in imem:  # x5 = u25(+1)
            imemap(x5)
    else:  # Si non: Indice terminal z_
        z_ = 1

    # Ordonnées positives et négatives
    if v_ == 0:
        v -= 6  # sq5 typ1
        y += 6  # sut typ1
```

```
                v5 -= 6  # sq5 typ5
                y5 += 6  # sut typ5
            if w_ == 0:
                w -= 6  # s25 typ1
                u += 1  # u
                w5 -= 6  # s25 typ5
            if z_ == 0:
                z -= 6  # sut typ1
                x += 6  # s25 typ1
                z5 -= 6  # sut typ5
                x5 += 6  # s25 typ5
            vwz = v_ + w_ + z_  # :vwz: Cumul des terminaux
            if vwz > 2:  # :vwz=3: Le tour est fait !
                break
    # Tableau :imem[ip(NP)]: Trier
    imem.sort()
    # 2- Copie (imem) sur tableau(itab)
    itab = copy.copy(imem)
    print('Nombres Premiers :ip:', imem)  # Affiche (les) NP
    # print('En {}'.format(time.time() - dod1))
    if u1 == 0:  # Zéro issue
        # Les multiplications NP produisent NC
        while 1:
            ion1 = len(itab)
            for i in imem:
                for y in itab:
                    i1 = y * i
                    if i1 not in itab \
                       and not uno % i1:
                        itab.append(i1)
                        # NC = Nombre Commun :itab[NC]: Résultat "ip"
            itab.sort()
            ion2 = len(itab)
            if ion1 == ion2:  # 4- :ion1::ion2: Motif de progression
                break
        for i in itab:  # Démultiplier :i: produire :d:
            d = uno // i
            ip, dv = i, d
            if ip < dv or ip == dv:  # Déblocage :uno = 2 ou 4:
                print('%s * %s  typ %s*%s' % (ip, dv, ip % 6, dv % 6))
    else:
        # Zéro issue :uno;onu:
        print('%s * %s  typ %s*%s' % (uno, onu, uno % 6, onu % 6))
        uno = 0

    print('Cosmic', uno, 'typ', uo6, 'long', len(itab))
    print('nphysic_7a.py En {}'.format(time.time() - dod0))


# Nombre original :onu:
onu = 99997
onuno(onu)
# nombre(onu): Charge zéro
# nombre(uno): Charge unité
```