TP_1-2 - CORRECTION

September 11, 2021

0.1 EXERCICE 3

[1]:
```
import math # import est une instruction, il n'y a pas de (...)
dir(math) # dir(...) est une fonction
```

[1]:
```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
```

```
 'isfinite',
 'isinf',
 'isnan',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'pi',
 'pow',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc']
```

[2]:
```
pi # 'pi' est un identificateur inconnu dans le noyau de base de Python
```

```
␣
↪---------------------------------------------------------------------------

NameError                                 Traceback (most recent call␣
↪last)

<ipython-input-2-b11676b257e2> in <module>
----> 1 pi # 'pi' est un identificateur inconnu dans le noyau de base de␣
↪Python


NameError: name 'pi' is not defined
```

[3]:
```
math.pi # on voit avec dir(math) que 'pi' figure dans le module math
print(math.cos(math.pi/3)) ; print(math.log10(2)) # chaque fonction doit être␣
 ↪précédée de math.
```

```
0.5000000000000001
0.3010299956639812
```

Noter l'erreur d'arrondi évidente sur $\cos \frac{\pi}{3}$, lié au calcul en virgule flottante avec mantisse finie.

[2]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
import math as m # après cela il faut utiliser 'm', le nom 'math' ne␣
↪fonctionnera pas
```

[4]:
```
print(m.pi) ; print(math.pi)
```

```
3.141592653589793


      ␣
↪---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call␣
↪last)

        <ipython-input-4-a3e72e82a197> in <module>
    ----> 1 print(m.pi) ; print(math.pi)


        NameError: name 'math' is not defined
```

[5]:
```
help(m.floor) ; help(m.ceil) # ou nom_de_fonction puis la touche F1
```

```
Help on built-in function floor in module math:

floor(x, /)
    Return the floor of x as an Integral.

    This is the largest integer <= x.

Help on built-in function ceil in module math:

ceil(x, /)
    Return the ceiling of x as an Integral.

    This is the smallest integer >= x.
```

[7]:
```
print(m.floor(1.2), m.floor(-1.2), m.ceil(1.2), m.ceil(-1.2))
```

```
1 -2 2 -1
```

[8]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
from math import *
help(radians) ; help(degrees) # plus besoin d'ouvrir 'math', car on a tout␣
↪importé
```

```
Help on built-in function radians in module math:
```

```
radians(x, /)
    Convert angle x from degrees to radians.

Help on built-in function degrees in module math:

degrees(x, /)
    Convert angle x from radians to degrees.
```

[9]:
```
print(radians(90)) ; print(degrees(m.pi/4)) # pas de pb, c'est bon
```

```
1.5707963267948966
45.0
```

[10]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
import math as m ; import cmath as cm ; import numpy as np
print(m.pi == np.pi) ; print(m.pi is np.pi)
```

```
True
False
```

Avec == sont comparées les *évaluations* de m.pi et np.pi, qui donnent le même flottant. Avec **is** sont comparés les objets informatiques, qui sont différents. On peut vérifier avec la fonction id(), on n'obtient pas les mêmes adresses :

[2]:
```
print(id(m.pi)) ; print(id(np.pi))
```

```
2188385966480
2188445306448
```

[5]:
```
print(m.cos(np.pi)) ; print(np.cos(m.pi))
```

```
-1.0
-1.0
```

Explication : avant de lancer la fonction m.cos ou np.cos, l'interpréteur évalue m.pi ou np.pi, et dans les deux cas l'évalue au même flottant, qui peut servir d'argument indifféremment aux deux fonctions.

[13]:
```
help(cm.polar) ; help(cm.phase) ; help(cm.rect) # conversion des usuelles d'un␣
↪complexe
```

```
Help on built-in function polar in module cmath:

polar(z, /)
    Convert a complex from rectangular coordinates to polar coordinates.

    r is the distance from 0 and phi the phase angle.

Help on built-in function phase in module cmath:
```

```
phase(z, /)
    Return argument, also known as the phase angle, of a complex.

Help on built-in function rect in module cmath:

rect(r, phi, /)
    Convert from polar coordinates to rectangular coordinates.
```

[21]:
```
cm.polar( 1/m.sqrt(2) + 1j / m.sqrt(2) )
```

[21]:
```
(0.9999999999999999, 0.7853981633974483)
```

[28]:
```
print(cm.phase( m.sqrt(3) / 2 + 1j / 2),'(radians)');
print(m.degrees( cm.phase(m.sqrt(3) / 2 + 1j / 2)),'(degrés)')
```

```
0.5235987755982989 (radians)
30.000000000000004 (degrés)
```

[30]:
```
cm.rect(2**0.5,cm.pi/4)
```

[30]:
```
(1.0000000000000002+1.0000000000000002j)
```

0.2 EXERCICE 4

[6]:
```
## fonction tanDeg avec le module math importé avec l'alias m
def tanDeg(x):
    return m.tan(x/180*m.pi) # on convertit la valeur x de degrés en radians␣
 ↪pour la fonction tangente
print(tanDeg(45))
```

```
0.9999999999999999
```

ce qui est correct aux erreurs d'arrondi près.

[8]:
```
## tester l'égalité de Pythagore entre 3 nombres triés
def testRec1(a,b,c):
    return c*c == b*b + a*a
print(testRec1(3,4,5)) ; print(testRec1(5,4,3)) # la fonction ne fontionne que␣
 ↪si c est le plus grand
```

```
True
False
```

[9]:
```
## tester l'égalité de Pythagore entre 3 nombres dans un ordre quelconque
def testRec2(a,b,c):
    return c*c == b*b + a*a or b*b == c*c + a*a or a*a == b*b + c*c
print(testRec2(3,4,5)) ; print(testRec2(5,4,3)) # la fonction fontionne dans␣
 ↪tous les cas
```

```
True
True
```

NB : ces tests d'égalité fonctionnent avec des entiers, mais si (a, b, c) peuvent être réels il faut tenir compte des problèmes d'arrondis et introduire une tolérance : $a^2 == b^2 + c^2$ sera remplacé par $|a^2 - b^2 - c^2| < \epsilon$ où ϵ est une "petite" quantité adaptée au contexte, par ex. 10^{-6} ou …

0.3 EXERCICE 6

[11]:
```
x = 11 # associe l'étiquette 'x' à l'entier 11
```

[12]:
```
y = x # 'x' est évalué à 11 avant affectation, donc cela équivaut à 'y = 11'
```

[14]:
```
print('x =',x,'et y =',y) # les deux étiquettes sont attachées à l'entier 11
```

```
x = 11 et y = 11
```

[15]:
```
x = 11 ; y = 22
z = x + y # équivaut à z = 33
x = y  # équivaut à x = 22
y = z # équivaut à y = 33
print(x+y+z) # donnera 88 car x, y et z sont évaluées à 22, 33, 33
```

```
88
```

[17]:
```
x = 11 ; x = x + 1 ; x+= 2 ; x *= 3 # utilisation de raccourcis Python
# les valeurs successives de x sont 11, 12, 14, 42(non affichées)
print('x =',x) # va afficher la dernière valeur soit 42
```

```
x = 42
```

[18]:
```
 x,y = 0,1 # affectation en parallèle
```

[20]:
```
x,y = y,x+y ; x,y = y,x+y ; x,y = y,x+y ; x,y = y,x+y
# équivaut à x,y = 1,1 puis x,y = 1,2 puis x,y = 2,3 puis x,y = 3,5
print('x =',x,'et y =',y) # va donc afficher ces dernières valeurs
```

```
x = 3 et y = 5
```

NB : ce procédé calcule les termes successifs de la *suite de Fibonacci*.

0.4 EXERCICE 7

[2]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
x = 11 ; y = 22 ; print('x =',x,'et y =',y)
```

```
x = 11 et y = 22
```

[3]:
```
z = x ; x = y ; y = z ; print('x =',x,'et y =',y)
```

```
x = 22 et y = 11
```

[1]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
x = 11 ; y = 22 ; print('x =',x,'et y =',y)
x, y = y, x ; print('x =',x,'et y =',y)
```

```
x = 11 et y = 22
x = 22 et y = 11
```

[1]:
```
# ne pas oublier de réinitialiser (redémarrer une nouvelle session)
x = 11 ; y = 22 ; print('x =',x,'et y =',y)
x = x + y ; y = x - y ; x = x - y ; print('x =',x,'et y =',y)
```

```
x = 11 et y = 22
x = 22 et y = 11
```

0.5 EXERCICE 8

[3]:
```
12 ; print(type(12)) ; print(id(12))
```

```
<class 'int'>
140736914170608
```

[4]:
```
Nb = 12 ; print(Nb) ; print(type(Nb)) ; print(id(Nb))
```

```
12
<class 'int'>
140736914170608
```

On constate que l'étiquette "Nb" a été associée à l'entier 12 qu'on reconnaît à son identifiant.

[5]:
```
Nb += 1 ; id(Nb) == id(1+3*4)
```

[5]: True

L'étiquette "Nb" est maintenant associée à l'entier 13 (évaluation de 1+3x4).

[6]:
```
y = x = Nb ; id(x), id(y)
```

[6]: (140736914170640, 140736914170640)

Les deux étiquettes "x" et "y" sont attachées au même entier 13 dont on a l'identifiant.

[7]:
```
 x *= 2 ; id(x), id(y)
```

[7]: (140736914171056, 140736914170640)

Les deux identifiants sont différents car l'étiquette "x" a été réassignée, elle est maintenant attachée à 26, alors que l'étiquette "y" est restée attaché à 13 ; on le vérifie par :

[8]:
```
id(26), id(13)
```

[8]: (140736914171056, 140736914170640)

0.6 EXERCICE 9

[1]:
```
pairs = [2, 4, 6, 8] ; impairs = [1, 3, 5, 7, 9]
```

[2]:
```
chiffres = pairs + impairs ; print(chiffres) # l'opérateur + réunit les deux␣
↪listes
```

```
[2, 4, 6, 8, 1, 3, 5, 7, 9]
```

[3]:
```
chiffres.sort() # ordonne la liste dans l'ordre croissant
chiffres.reverse() # inverse l'ordre des éléments
print(chiffres) # car ces méthodes modifient la liste mais n'affichent rien
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

[4]:
```
chiffres.append(0) # méthode importante pour ajouter un élément (ou chiffres =␣
↪chiffres +[0])
print(chiffres)
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

On veut récupérer le chiffre '5' ; pour cela on peut… :

- réfléchir à sa position et lire le bon élément de la liste

[15]:
```
chiffres[4] # ne pas oublier que les indices commencent à 0
```

[15]:
```
5
```

- faire rechercher par 'index' la position de '5' dans la liste puis lire :

[16]:
```
chiffres[chiffres.index(5)] # car chiffres.index(5) renvoie 4
```

[16]:
```
5
```

On peut utiliser la méthode des "listes en compréhension" :

[5]:
```
chiffres = [k for k in chiffres if k <= 6] ; chiffres
```

[5]:
```
[6, 5, 4, 3, 2, 1, 0]
```

[6]:
```
chiffres = [k for k in chiffres if k%2 != 0] ; chiffres
```

[6]:
```
[5, 3, 1]
```

[7]:
```
chiffres.append('bravo') ; chiffres
```

[7]:
```
[5, 3, 1, 'bravo']
```

[8]:
```
chiffres = chiffres + ['super'] ; chiffres
```

[8]:
```
[5, 3, 1, 'bravo', 'super']
```

[10]:
```
chiffres[len(chiffres) - 1] = 'trop fort' # ou chiffres[-1] = 'trop fort'
chiffres
```

[10]:
```
[5, 3, 1, 'bravo', 'trop fort']
```

[13]:
```
chiffres.pop() # pop renvoie le dernier élément...
```

[13]:
```
'trop fort'
```

[14]:
```
chiffres # et l'enlève de la liste !
```

[14]:
```
[5, 3, 1, 'bravo']
```

0.7 EXERCICE 10

[1]:
```
siecle21 = list(range(2001,2101)) ; siecle21 # vérifier début 2001 et fin 2100 !
```

[1]:
```
[2001,
 2002,
 2003,
 2004,
 2005,
 2006,
 2007,
 2008,
 2009,
 2010,
 2011,
 2012,
 2013,
 2014,
 2015,
 2016,
 2017,
 2018,
 2019,
 2020,
 2021,
 2022,
 2023,
 2024,
 2025,
 2026,
 2027,
 2028,
 2029,
 2030,
 2031,
```

2032,
2033,
2034,
2035,
2036,
2037,
2038,
2039,
2040,
2041,
2042,
2043,
2044,
2045,
2046,
2047,
2048,
2049,
2050,
2051,
2052,
2053,
2054,
2055,
2056,
2057,
2058,
2059,
2060,
2061,
2062,
2063,
2064,
2065,
2066,
2067,
2068,
2069,
2070,
2071,
2072,
2073,
2074,
2075,
2076,
2077,
2078,

```
2079,
2080,
2081,
2082,
2083,
2084,
2085,
2086,
2087,
2088,
2089,
2090,
2091,
2092,
2093,
2094,
2095,
2096,
2097,
2098,
2099,
2100]
```

[17]:
```
table = [ (n1, n2, n1*n2) for n1 in range(1,10) for n2 in range(1,10)]
table
```

[17]:
```
[(1, 1, 1),
 (1, 2, 2),
 (1, 3, 3),
 (1, 4, 4),
 (1, 5, 5),
 (1, 6, 6),
 (1, 7, 7),
 (1, 8, 8),
 (1, 9, 9),
 (2, 1, 2),
 (2, 2, 4),
 (2, 3, 6),
 (2, 4, 8),
 (2, 5, 10),
 (2, 6, 12),
 (2, 7, 14),
 (2, 8, 16),
 (2, 9, 18),
 (3, 1, 3),
 (3, 2, 6),
 (3, 3, 9),
```

```
(3, 4, 12),
(3, 5, 15),
(3, 6, 18),
(3, 7, 21),
(3, 8, 24),
(3, 9, 27),
(4, 1, 4),
(4, 2, 8),
(4, 3, 12),
(4, 4, 16),
(4, 5, 20),
(4, 6, 24),
(4, 7, 28),
(4, 8, 32),
(4, 9, 36),
(5, 1, 5),
(5, 2, 10),
(5, 3, 15),
(5, 4, 20),
(5, 5, 25),
(5, 6, 30),
(5, 7, 35),
(5, 8, 40),
(5, 9, 45),
(6, 1, 6),
(6, 2, 12),
(6, 3, 18),
(6, 4, 24),
(6, 5, 30),
(6, 6, 36),
(6, 7, 42),
(6, 8, 48),
(6, 9, 54),
(7, 1, 7),
(7, 2, 14),
(7, 3, 21),
(7, 4, 28),
(7, 5, 35),
(7, 6, 42),
(7, 7, 49),
(7, 8, 56),
(7, 9, 63),
(8, 1, 8),
(8, 2, 16),
(8, 3, 24),
(8, 4, 32),
(8, 5, 40),
```

```
 (8, 6, 48),
 (8, 7, 56),
 (8, 8, 64),
 (8, 9, 72),
 (9, 1, 9),
 (9, 2, 18),
 (9, 3, 27),
 (9, 4, 36),
 (9, 5, 45),
 (9, 6, 54),
 (9, 7, 63),
 (9, 8, 72),
 (9, 9, 81)]
```

[18]:
```
div16 = [i for i in siecle21 if i%16 == 0] ; div16
```

[18]:
```
[2016, 2032, 2048, 2064, 2080, 2096]
```

[21]:
```
siecle21 = [k for k in siecle21 if k not in div16 ] ; siecle21 # on vérifie que␣
↪ça marche
```

[21]:
```
[2001,
 2002,
 2003,
 2004,
 2005,
 2006,
 2007,
 2008,
 2009,
 2010,
 2011,
 2012,
 2013,
 2014,
 2015,
 2017,
 2018,
 2019,
 2020,
 2021,
 2022,
 2023,
 2024,
 2025,
 2026,
 2027,
```

2028,
2029,
2030,
2031,
2033,
2034,
2035,
2036,
2037,
2038,
2039,
2040,
2041,
2042,
2043,
2044,
2045,
2046,
2047,
2049,
2050,
2051,
2052,
2053,
2054,
2055,
2056,
2057,
2058,
2059,
2060,
2061,
2062,
2063,
2065,
2066,
2067,
2068,
2069,
2070,
2071,
2072,
2073,
2074,
2075,
2076,
2077,

```
 2078,
 2079,
 2081,
 2082,
 2083,
 2084,
 2085,
 2086,
 2087,
 2088,
 2089,
 2090,
 2091,
 2092,
 2093,
 2094,
 2095,
 2097,
 2098,
 2099,
 2100]
```

Une année qui n'est pas multiple de 100 est bissextile si c'est un multiple de 4, et si c'est un nombre entier de siècles elle n'est bissextile que si ce nombre de siècles est multiple de 4 ; *il y a donc 97 années bisextiles sur 400*, d'où la durée moyenne de l'année : 365 + 97/400 = 365,2425 jours solaires.

[4]:
```
bis = [a for a in range(1583,2022) if (a%4 == 0 and a%100 !=0) or (a%100 == 0
 ↪and (a//100)%4 == 0)]
print('Il y a eu',len(bis),'années bissextiles depuis la réforme grégorienne.')
print('Ce sont:',bis)
```

```
Il y a eu 107 années bissextiles depuis la réforme grégorienne.
Ce sont: [1584, 1588, 1592, 1596, 1600, 1604, 1608, 1612, 1616, 1620, 1624,
1628, 1632, 1636, 1640, 1644, 1648, 1652, 1656, 1660, 1664, 1668, 1672, 1676,
1680, 1684, 1688, 1692, 1696, 1704, 1708, 1712, 1716, 1720, 1724, 1728, 1732,
1736, 1740, 1744, 1748, 1752, 1756, 1760, 1764, 1768, 1772, 1776, 1780, 1784,
1788, 1792, 1796, 1804, 1808, 1812, 1816, 1820, 1824, 1828, 1832, 1836, 1840,
1844, 1848, 1852, 1856, 1860, 1864, 1868, 1872, 1876, 1880, 1884, 1888, 1892,
1896, 1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936, 1940, 1944, 1948,
1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992, 1996, 2000,
2004, 2008, 2012, 2016, 2020]
```

[8]:
```
mul7 = [k for k in range(1, 101) if k%7 == 0] ; print(mul7)
mul7 = [7*k for k in range(1, 100//7+1)] ; print(mul7)
mul7 = list(range(7, 101,7)) ; print(mul7)
```

```
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98]
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98]
```

```
[7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98]
```

[9]:
```
carres = [k*k for k in range(1,101)] # car cela donne tous les carrés jusqu'à␣
↪10 000
print(carres)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324,
361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961, 1024, 1089,
1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849, 1936, 2025, 2116,
2209, 2304, 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136, 3249, 3364, 3481,
3600, 3721, 3844, 3969, 4096, 4225, 4356, 4489, 4624, 4761, 4900, 5041, 5184,
5329, 5476, 5625, 5776, 5929, 6084, 6241, 6400, 6561, 6724, 6889, 7056, 7225,
7396, 7569, 7744, 7921, 8100, 8281, 8464, 8649, 8836, 9025, 9216, 9409, 9604,
9801, 10000]
```

[13]:
```
carres9 = [n for n in carres if n%10 == 9]
print(carres9)
print('Il y en a', len(carres9),'.')
```

```
[9, 49, 169, 289, 529, 729, 1089, 1369, 1849, 2209, 2809, 3249, 3969, 4489,
5329, 5929, 6889, 7569, 8649, 9409]
Il y en a 20 .
```

[19]:
```
cour1 = ['Alex', 'Bob', 'Charlie', 'David', 'Eddy', 'Franck'] ; clas1 = [2, 4,␣
↪1, 6, 3, 5]
```

[20]:
```
premier = cour1[clas1.index(1)] ; print(premier)
```

```
Charlie
```

[21]:
```
dernier = cour1[clas1.index(max(clas1))] ; print(dernier) # ça marche pour tout␣
↪nombre de coureurs
```

```
David
```

[22]:
```
del(cour1[clas1.index(max(clas1))])
del(clas1[clas1.index(max(clas1))])
print(cour1, clas1)
```

```
['Alex', 'Bob', 'Charlie', 'Eddy', 'Franck'] [2, 4, 1, 3, 5]
```

[23]:
```
tup = [( cour1[clas1.index(rang)], rang ) for rang in clas1] ; tup
```

[23]:
```
[('Alex', 2), ('Bob', 4), ('Charlie', 1), ('Eddy', 3), ('Franck', 5)]
```

[26]:
```
clas2 = list.copy(clas1) ; clas2.sort()
cour2 = [cour1[clas1.index(k)] for k in clas2]
cour2, clas2
```

[26]:
```
(['Charlie', 'Alex', 'Eddy', 'Bob', 'Franck'], [1, 2, 3, 4, 5])
```

0.8 EXERCICE 11

[22]:
```
def resout(a,b):
    if a == 0:
        print('a ne doit pas être nul !')
        return None
    else:
        return -b/a
resout(0,3) ; resout(2,3)
```

```
a ne doit pas être nul !
```

[22]:
```
-1.5
```

on peut vérifier que le premier appel de la fonction renvoie bien 'None' :

[23]:
```
print(resout(0,3))
```

```
a ne doit pas être nul !
None
```

0.9 EXERCICE 12

[24]:
```
def sign(x):
    if x == 0:
        return 0
    elif x > 0:
        return 1
    else:
        return -1
sign(-3), sign(0), sign(3) # cela définit un triplet (de la classe 'tuple')
```

[24]:
```
(-1, 0, 1)
```

0.10 EXERCICE 13

[26]:
```
for i in range(3,10): # prendra les valeurs de 3 à 9
    print("numéro",i) # affichera donc sur 7 lignes ces valeurs
print('i =',i) # à la fin de la boucle la variable i conservera la denrière␣
 ↪valeur soit 9
```

```
numéro 3
numéro 4
numéro 5
numéro 6
numéro 7
numéro 8
numéro 9
i = 9
```

[27]:
```
def f(start,stop):
    for j in range(start,stop): # la différence : j est ici une variable locale
        print("numéro",j)
f(3,10) ; print('j =',j) # on n'est plus dans l'exécution de la fonction, j est␣
 ↪inconnue !
```

```
numéro 3
numéro 4
numéro 5
numéro 6
numéro 7
numéro 8
numéro 9
```

```
      ␣
↪---------------------------------------------------------------------------

       NameError                                 Traceback (most recent call␣
↪last)

       <ipython-input-27-986c390aefde> in <module>
         2     for j in range(start,stop): # la différence : j est ici une␣
↪variable locale
         3         print("numéro",j)
   ----> 4 f(3,10) ; print('j =',j) # on n'est plus dans l'exécution de la␣
↪fonction, j est inconnue !

       NameError: name 'j' is not defined
```

[29]:
```
for i in range(3,10):
    print("numéro"+i)
```

```
      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-29-b916e44e2fd3> in <module>
         1 for i in range(3,10):
   ----> 2     print("numéro"+i)

       TypeError: can only concatenate str (not "int") to str
```

L'opérateur + ne peut réunir une chaîne et un entier, mais il peut réunir (concaténer) deux chaînes ; nous pouvons le vérifier en convertissant l'entier i e nchaîne de caractère par la fonction str() :

[30]:
```
for i in range(3,10):
    print("numéro"+str(i)) # pour que ce soit plus joli il faudrait insérer une␣
 ↪espace
```

```
numéro3
numéro4
numéro5
numéro6
numéro7
numéro8
numéro9
```

[31]:
```
somme = 0 # c'est une initialisation
for i in range(1,11): # i prendra les valeurs de 1 à 10
    somme += i # ces valeurs seront l'une après l'autre ajoutées à somme
print(somme) # qui vaudra donc 55 en sortie de boucle
```

```
55
```

Rappel : la somme des entiers naturels de 1 à N vaut N(N+1)/2. Utilisation de la fonction 'sum':

[33]:
```
print(sum(i for i in range(1,11)))
```

```
55
```

0.11 EXERCICE 14

[35]:
```
somme = 0 ; fin = 11
for i in range(1,fin): # i prendra les valeurs de 1 à fin-1
    if i %2 != 0: # sélection des nombres impairs
        somme += i # dont on fait la somme
print(somme)
```

```
100
```

Tiens, c'est un carré ! Recommençons avec fin = 8, on trouve 16 ; avec fin = 20 on trouve 100... C'est que la somme des N premiers nombres impairs donne N^2 (voir (sur internet par exemple) une jolie preuve géométrique avec des "équerres" qui s'emboîtent pour former un carré).

Ce serait plus simple à utiliser avec une fonction d'argument fin :

[37]:
```
def somImp(fin):
    somme = 0 # c'est une variable locale
    for i in range(1,fin,2): # on contruit directement les nombres impairs
        somme += i
    return somme
print(somImp(20))
```

100

0.12 EXERCICE 15

[40]:
```
def facto(nb):
    f = 1 # contiendra la factorielle en cours de calcul
    x = nb # variable locale dans laquelle on recopie la valeur de départ
    if x == 0:
        return 1 # et c'est fini
    else:
        while x > 0:
            f = f * x
            x -= 1 # décrémentation de x qui finira donc par arriver à 0
    return f
print('5! =',facto(5)) ; print('10! =',facto(10))
```

```
5! = 120
10! = 3628800
```

Noter que la variable de travail dans laquelle se construit progressivement le résultat est initialisée à l'élément neutre de l'opération réalisée : précédemment c'était 0 pour une somme, maintenant c'est 1 pour un produit.

0.13 EXERCICE 16

[49]:
```
def descend(n):
    x = n ; c = 0 # c comptera les étapes
    while x > 1:
        c += 1
        if x % 2 == 0:
            x /= 2
        else:
            x -= 1
    return c
n = 123456789
print('Pour',n,'on arrive à 1 après',descend(n),'étapes.')
```

```
Pour 123456789 on arrive à 1 après 41 étapes.
```

On divise un nombre par 2 en une ou deux étapes, selon qu'il est pair ou impair. On vérifie qu'il faut 10 étapes en partant de 1024 = 2^10, mais il en faut 18 en partant de 1023. Disons qu'en moyenne il faut ajouter une quinzaine d'étapes chaque fois qu'on multiplie le nombre de départ d'un facteur 1000.

[52]:
```
n = 1234 ; print('Pour',n,'on arrive à 1 après',descend(n),'étapes.')
n = 1234 * 5678 * 910 ; print('Pour',n,'on arrive à 1 après',descend(n),'étapes.
 ↪')
```

```
Pour 1234 on arrive à 1 après 14 étapes.
Pour 6376053320 on arrive à 1 après 46 étapes.
```

0.14 EXERCICE 17

Calculer un nombre de termes déterminé => boucle 'for'. Calculer les termes inférieurs à une valeur => on ne sait pas quand on y arrivera => boucle 'while'.

[54]:
```
def fibo(max): # on se donnera max > 0
    L = [0,1] # initialisée avec F0, F1
    while L[-1] <= max: # L[-1] désigne le dernier élément de L ; équivaut à␣
 ↪L[len(L)-1]
        L.append(L[-1] + L[-2]) # ou L[len(L)-1] + L[len(L)-2]
    L.pop() # ou L = L[1:len(L)-2] pour enlever le dernier élément qui est trop␣
 ↪grand
    return L
```

NB: - lorsque la boucle s'arrête, c'est que le dernier terme calculé est devenu supérieur à max, il faut donc l'enlever pour répondre à la question posée ! - l'énoncé ne dit pas si on veut un dernier terme < M ou <= à M ; on a choisi <= ici.

[55]:
```
fibo(1000)
```

[55]:
```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987]
```

[57]:
```
L1 = fibo(1000) ; seq = range(2,len(L1)) # seq contiendra les entiers de 2 à␣
 ↪len(L)-1
L2 = [ L1[k] / L1[k-1] for k in seq ] # il faut éliminer F0 = 0 du calcul
print(L2)
```

```
[1.0, 2.0, 1.5, 1.6666666666666667, 1.6, 1.625, 1.6153846153846154,
1.619047619047619, 1.6176470588235294, 1.6181818181818182, 1.6179775280898876,
1.6180555555555556, 1.6180257510729614, 1.6180371352785146, 1.618032786885246]
```

Ces valeurs semblent se rapprocher du nombre d'or $\varphi \approx 1{,}618\ 034$. C'est bien le cas, car si on admet que le rapport F_n/F_{n-1} tend vers une limite x, la relation de récurrence $F_n = F_{n-1} + F_{n-2}$ donne pour n grand :

$$x * x * F_{n-2} = x * F_{n-2} + F_{n-2} \text{ soit } x^2 = x + 1, cqfd.$$

0.15 EXERCICE 18

[32]:
```
def isprime(N): # on ne contrôlera pas ici la valeur donnée par l'utilisateur
    fin = N**.5 // 1 # équivaut à math.floor(N) mais évite de charger math
    D = 2 # initialisation du diviseur éventuel
    flag = True # on utilise un drapeau qu'on baissera quand il faudra arrêter
    while flag and D <= fin:
        flag = N%D !=0
        D += 1
    if flag:
```

```
        print(N,'est premier.')
    else:
        print(N,'est divisible par',D-1)
    return flag
isprime(99), isprime(100), isprime(101)
```

```
99 est divisible par 3
100 est divisible par 2
101 est premier.
```

[32]:
```
(False, False, True)
```

Noter la différence entre l'effet des 'print' et ce que renvoie la fonction. NB: on pourrait traiter le cas de 2 à part et ne tester que les diviseurs impairs, cela économiserait la moitié des calculs ; cela sera fait dans les fonctions suivantes.

[36]:
```
def isprime2(N): # sans les affichages
    fin = N**.5 // 1
    D = 3 # on traitera 2 à part dans la fonction qui appellera isprime2
    flag = True
    while flag and D <= fin:
        flag = N%D !=0
        D += 2 # on ne testera donc que des diviseurs impairs
    return flag

def listprime(N): # on suppose N entier > 1
    L = [2] # initialisation de la liste
    if N == 2: # élimination du cas particulier
        return L
    else:
        x = 3 # le premier candidat nombre premier > 2
        while L[len(L)-1] <= N: # ou L[-1] <= N
            if isprime2(x):
                L.append(x)
            x += 2 # de façon à ne tester que les entiers impairs
        L.pop()
    return L

listprime(100)
```

[36]:
```
[2,
 3,
 5,
 7,
 11,
 13,
 17,
 19,
```

```
23,
29,
31,
37,
41,
43,
47,
53,
59,
61,
67,
71,
73,
79,
83,
89,
97]
```

Remarquons encore une amélioration possible: au lieu de chercher le diviseur parmi tous les entiers impairs entre 3 et sqrt(N), on pourrait ne chercher que parmi les nombres premiers qu'on a déjà.