new idea

SOLUTION

Diogenes Game $O(n \log n), O(n)$

Algorithm 1 FINDHONESTPERSON(A)

Input: $|A| = 2^k$ for $k \geq 0$
Output: An honest person in A

```
1: if |A| = 1 then
2:     return A[1]
3: end if
4: (A_1, A_2) ← split(A)
5: p_1 ← FINDHONESTPERSON(A_1)
6: p_2 ← FINDHONESTPERSON(A_2)
7: for x ∈ A \ {p_1} do
8:     if x calls p_1 dishonest then
9:         return p_2
10:    end if
11: end for
12: return p_1
```

$O(n \log n)$

$O(n \log n)$

```
def computeHonestPerson_MEDIUM_RECURSIVE(A, isHonest):
    n = len(A)

    if n == 1: return A[0]

    p1 = computeHonestPerson_MEDIUM_RECURSIVE(A[:n//2], isHonest)
    p2 = computeHonestPerson_MEDIUM_RECURSIVE(A[n//2:], isHonest)

    count1 = 0
    for i in range(0, len(A)):
        if A[i] != p1 and isHonest(A[i], p1):
            count1 += 1

    if 2*count1 >= n-1:
        return p1
    else:
        return p2
```

Algorithm 1 TrouverPersonneHonnête(A)

Input: A — une liste de n personnes
Output: Une personne honnête dans A

1: $B \leftarrow []$
2: **for** $i \leftarrow 0$ **to** $n-1$ **do**
3: **if** B est vide **then**
4: Ajouter $A[i]$ à B
5: **else**
6: Interroger $A[i]$ à propos de $B[-1]$
7: **if** $A[i]$ dit que $B[-1]$ est honnête **then**
8: Ajouter $A[i]$ à B
9: **else**
10: Retirer $B[-1]$ de B
11: **end if**
12: **end if**
13: **end for**
14: **return** $B[0]$

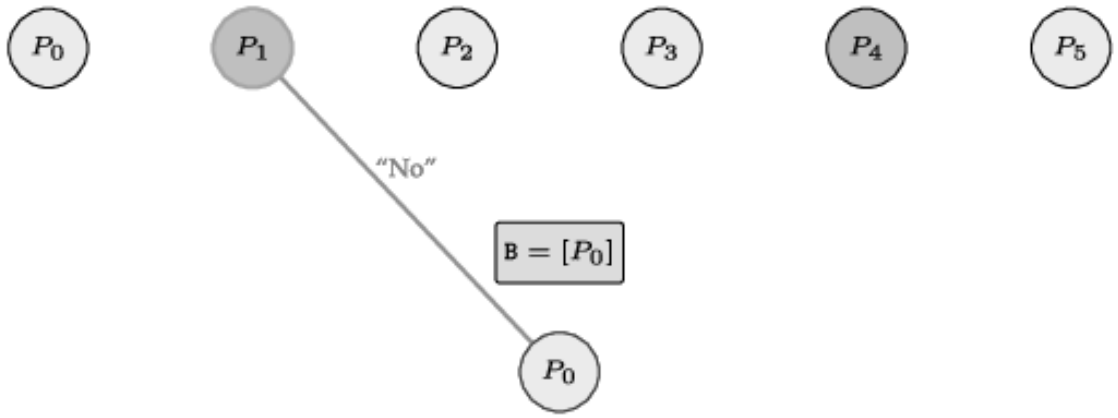
$O(n)$

$O(n)$

```
def computeHonestPerson_FAST_CHAIN(A, isHonest):
    n = len(A)

    B = []
    for i in range(0, n):
        if len(B) == 0:
            B.append(A[i])
        else:
            if isHonest(A[i], B[-1]):
                B.append(A[i])
            else:
                B.pop()
    return B[ 0 ]
```

i = 0: Add P_0 to B

$B = [P_0]$



i = 1: Remove P_0 from B and do not add P_1 to B

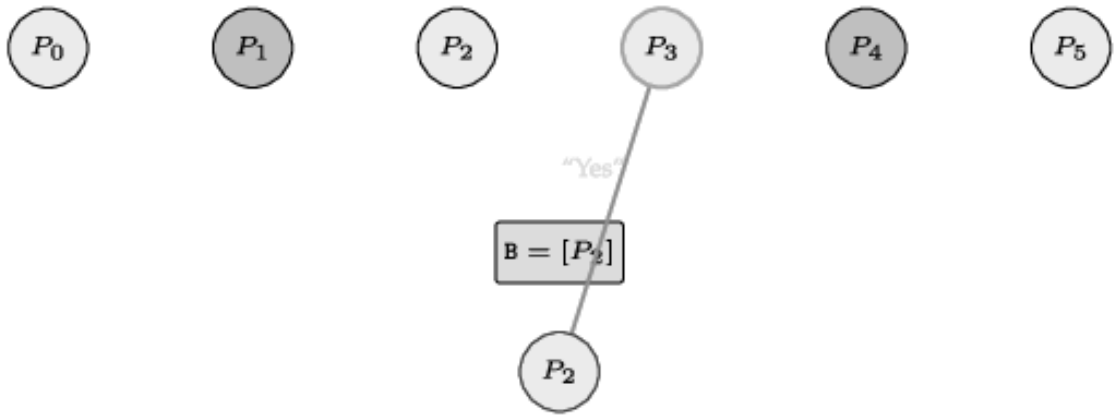
B = []

B = []

B empty → add P_2

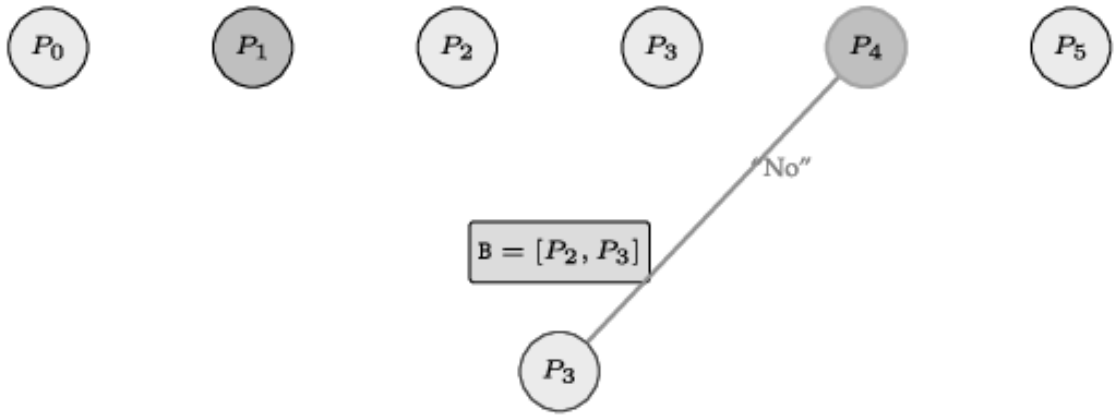
i = 2: Add P_2 to B

B = [P_2]



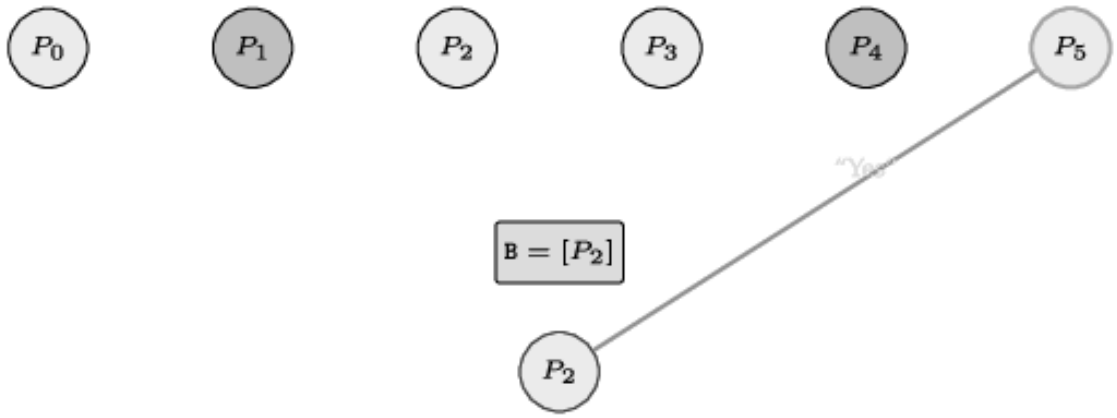
i = 3: Add P_3 to B

$\texttt{B} = [P_2, P_3]$



i = 4: Remove P_3 from B and do not add P_4 to B

$\texttt{B} = [P_2]$



i = 5: Add P_5 to $\texttt{B}$

$$\boxed{\texttt{B} = [P_2, P_5]}$$

$$\boxed{\texttt{B} = [P_2, P_5]}$$

Return $\texttt{B}[0] = P_2$ (Honest)

new idea

MODULES

Les modules sont des programmes Python qui contiennent des fonctions

on les appelle aussi bibliothèques ou libraries

Ce sont des "boîtes à outils" qui vont vous être très utiles.

Les développeurs de Python ont mis au point de nombreux modules qui effectuent une quantité phénoménale de tâches.

new idea

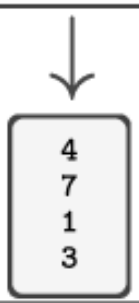
MODULES: RANDOM

`module` random: générer des variables aléatoires

l'instruction import donne accès à **toutes** les fonctions du module random.

nous utilisons la fonction randint du module random

```
import random

x = random.randint(0,10)
print(x)
x = random.randint(0,10)
print(x)
x = random.randint(0,10)
print(x)
x = random.randint(0,10)
print(x)
```



module random: générer des variables aléatoires

l'instruction import donne accès à **toutes** les fonctions du module random.

```
import random

x = random.choice([3,5, "hello", "bye", 3.51, False])
print(x)
x = random.choice([3,5, "hello", "bye", 3.51, False])
print(x)
x = random.choice([3,5, "hello", "bye", 3.51, False])
print(x)
```

choice(A): renvoie un élément aléatoire de la séquence

```
False
hello
5
```

module random: générer des variables aléatoires

random — Génère des nombres pseudo-aléatoires

Code source : Lib/random.py

Ce module implémente des générateurs de nombres pseudo-aléatoires pour différentes distributions.

Pour les entiers, il existe une sélection uniforme à partir d'une plage. Pour les séquences, il existe une sélection uniforme d'un élément aléatoire, une fonction pour générer une permutation aléatoire d'une liste sur place et une fonction pour un échantillonnage aléatoire sans remplacement.

Pour l'ensemble des réels, il y a des fonctions pour calculer des distributions uniformes, normales (gaussiennes), log-normales, exponentielles négatives, gamma et bêta. Pour générer des distributions d'angles, la distribution de *von Mises* est disponible.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the half-open range `0.0 <= X < 1.0`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

`module` random: générer des variables aléatoires

À l'aide du mot-clé from, on peut importer une fonction spécifique d'un module donné.

```
from random import randint

x = randint(0, 100)
print(x)
```

```
62
```

`module` random: générer des variables aléatoires

À l'aide du mot-clé from, on peut importer une fonction spécifique d'un module donné.

```
from random import randint

x = randint(0, 100)
print(x)

x = random.choice([3,6,1,6])
print(x)
```

```
90
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 6, in <module>
    x = random.choice([3,6,1,6])
        ^^^^^^
NameError: name 'random' is not defined. Did you forget to import 'random'?
```

module random: générer des variables aléatoires

tout

```
from random import *

x = randint(0, 100)
print(x)

x = choice([3,6,1,6])
print(x)
```

```
18
6
```

module random: générer des variables aléatoires

```
from random import *

x = random.randint(0, 100)
print(x)

x = random.choice([3,6,1,6])
print(x)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    x = random.randint(0, 100)
        ^^^^^^^^^^^^^^
AttributeError: 'builtin_function_or_method'
object has no attribute 'randint'
```

ils sont tous équivalents

```
from random import *

x = randint(0, 100)
print(x)

x = choice([3,6,1,6])
print(x)
```

```
import random

x = random.randint(0, 100)
print(x)

x = random.choice([3,6,1,6])
print(x)
```

```
import random as mynameforrandom

x = mynameforrandom.randint(0, 100)
print(x)

x = mynameforrandom.choice([3,6,1,6])
print(x)
```

random.choices(A, k = 3): renvoie une liste de 3 éléments aléatoire de la séquence `A`

```
import random

X = random.choices([3,5, "hello", "bye", 3.51, False], k = 1)
print(X)

X = random.choices([3,5, "hello", "bye", 3.51, False], k = 5)
print(X)

X = random.choices([3,5, "hello", "bye", 3.51, False], k = 10)
print(X)
```

```
['hello']
[3, 5, False, 5, 'bye']
[False, False, 3.51, 'bye', 3, 'hello', 5, 'bye', 'hello', 'hello']
```

new idea

MODULES: SYS

module sys: des fonctions et des variables spécifiques à Python

```
import sys

print(type(sys.argv))
print(sys.argv)
```

une liste globale toujours présente !

```
python3 t.py a fasdf 425 asdfasdf
```

```
<class 'list'>
['t.py', 'a', 'fasdf', '425', 'asdfasdf']
```

fichier python

1er argument

`module` sys: des fonctions et des variables spécifiques à Python

sys — Paramètres et fonctions propres à des systèmes

Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier. Ce module est toujours disponible.

sys.abiflags

Contient, sur les systèmes POSIX où Python a été compilé avec le script `configure`, les *ABI flags* tels que définis par la PEP 3149.

Nouveau dans la version 3.2.

Modifié dans la version 3.8: Les options par défaut sont devenues des chaines de caractères vides (l'option `m` pour *pymalloc* a été enlevée).

Disponibilité : Unix.

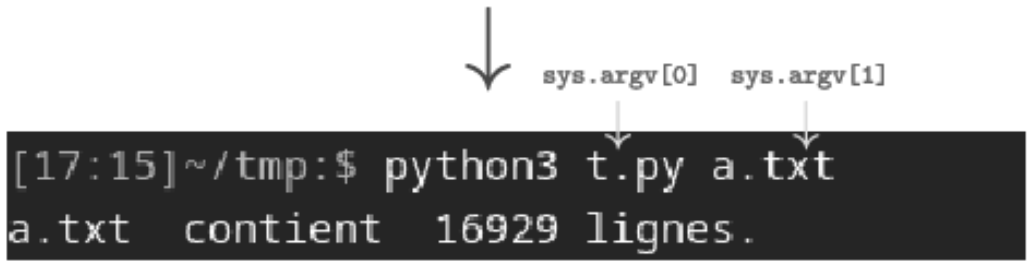
sys.addaudithook(*hook*)

Append the callable *hook* to the list of active auditing hooks for the current (sub)interpreter.

Quand un événement d'audit est déclenché par la fonction `sys.audit()`, chaque point d'entrée est appelé dans l'ordre dans lequel il a été ajouté avec le nom de l'événement et le *n*-uplet des arguments. Les points d'entrées qui sont ajoutés par `PySys_AddAuditHook()` sont appelés les premiers, suivi par les fonctions de rappel ajoutées dans l'interpréteur en cours d'exécution. Les points d'entrées peuvent *logger* l'événement, lever une exception pour stopper l'opération ou terminer le processus entièrement.

`module` sys: des fonctions et des variables spécifiques à Python

```
import sys

if len(sys.argv) != 2:
    sys.exit("ERREUR : il faut exactement un argument.")

nom_fichier = sys.argv[1]
taille = 0

inputfile = open(nom_fichier, "r")
taille = len(inputfile.readlines())

print(nom_fichier, " contient ", taille, "lignes.")
```



new idea

MODULES: OS

`module` os: gère l'interface avec le système d'exploitation.

```
import os

if os.path.exists("a.txt"):
    print("a.txt is there :)")
else:
    print("a.txt NOT found :( ")
```

```
a.txt is there :)
```

`module` os: gère l'interface avec le système d'exploitation.

```
import os

filepath = os.getcwd()

print(filepath)

L = os.listdir()

print(L)
```

```
/home/nabil/tmp
['LPduality.tex', 'independent_set_planar_graphs.tex',
      'lecture11c-handout.pdf', 'lecture12b-handout.
    pdf', 'notes.txt', 't.pdf', 't.log', 't.aux', 't.
    pytxcode', 't.pyg', 't.tex', 'a.txt', 't.py']
```

`module` os: gère l'interface avec le système d'exploitation.

os — Diverses interfaces pour le système d'exploitation

Code source: Lib/os.py

Ce module fournit une façon portable d'utiliser les fonctionnalités dépendantes du système d'exploitation. Si vous voulez simplement lire ou écrire un fichier, voir `open()`, si vous voulez manipuler les chemins de fichiers, voir le module `os.path`, et si vous voulez lire toutes les lignes de tous les fichiers de la ligne de commande, voir le module `fileinput`. Pour la création de fichiers et de répertoires temporaires, voir le module `tempfile`, et pour la manipulation de fichiers et de répertoires de haut niveau, voir le module `shutil`.

Notes sur la disponibilité de ces fonctions :

- La conception des modules natifs Python dépendants du système d'exploitation est qu'une même fonctionnalité utilise une même interface. Par exemple, la fonction `os.stat(path)` renvoie des informations sur les statistiques de *path* dans le même format (qui est originaire de l'interface POSIX).
- Les extensions propres à un certain système d'exploitation sont également disponible par le module `os`, mais les utiliser est bien entendu une menace pour la portabilité.
- Toutes les fonctions acceptant les chemins ou noms de fichiers acceptent aussi bien des *bytes* que des *string*, et si un chemin ou un nom de fichier est renvoyé, il sera du même type.
- On VxWorks, os.popen, os.fork, os.execv and os.spawn*p* are not supported.
- On WebAssembly platforms `wasm32-emscripten` and `wasm32-wasi`, large parts of the `os` module are not available or behave differently. API related to processes (e.g. `fork()`, `execve()`), signals (e.g. `kill()`, `wait()`), and resources (e.g. `nice()`) are not available. Others like `getuid()` and `getpid()` are emulated or stubs.

new idea

MODULES: TIME

```
import time
print( time.time() )
```

```
1743081743.435801
```

```
import time
n = pow(10,5)
start_time = time.time()
sum = 0
for i in range(0, n):
    sum += i
end_time = time.time()
print("time: ", end_time - start_time)
```

```
time:  0.0070209503173828125
```

```
import time
n = pow(10,6)
start_time = time.time()
sum = 0
for i in range(0, n):
    sum += i
end_time = time.time()
print("time: ", end_time - start_time)
```

```
time:  0.05660414695739746
```

```
import time
n = pow(10,7)
start_time = time.time()
sum = 0
for i in range(0, n):
    sum += i
end_time = time.time()
print("time: ", end_time - start_time)
```

```
time:  0.5137147903442383
```

new idea

Numpy Vectors

```
import numpy as np

A = np.array([1, 2.2, 34, 51])

print( A )

print( type(A) )

print( A.size )
```

```
[1.  2.2 34.  51.]
<class 'numpy.ndarray'>
4
```

```
import numpy as np

A = np.array([1.0, 2.2, 34, 51])

print(A)
print(type(A))

B = list(A)

print(B)
print(type(B))

C = np.array(B)

print(C)
print(type(C))
```

```
[ 1.   2.2 34.  51. ]
<class 'numpy.ndarray'>
[1.0, 2.2, 34.0, 51.0]
<class 'list'>
[ 1.   2.2 34.  51. ]
<class 'numpy.ndarray'>
```

new idea

Numpy Vectors

Operations

```
import numpy as np

A = np.array( [2,1,5] )

l = np.linalg.norm(A)
print(l)

B = 2 * A
print(B)
```

$\|A\|_2 = \sqrt{2^2 + 1^2 + 5^2} = 5.477...$ ↓

```
5.477225575051661
[ 4  2 10]
```

```python
import numpy as np

A = np.array( [2,1,5] )
B = np.array( [56,42,3] )

print(A+B, "\n")
print(A*B, "\n")
print(A/B)
```

Les opérations arithmétiques classiques ($+$, $-$, $*$, $/$, $**$) s'appliquent **élément par élément** entre tableaux de mêmes dimensions

```
[58 43  8]

[112  42  15]

[0.03571429 0.02380952 1.66666667]
```

```python
import numpy as np

A = np.array( [2,1,5] )
B = np.array( [56,42,3] )


D = np.dot(A,B)
print(D)
```

169

$$\mathbf{A} \cdot \mathbf{B} = 2 * 56 + 1 * 42 + 5 * 3 = 169$$

new idea

Numpy Vectors

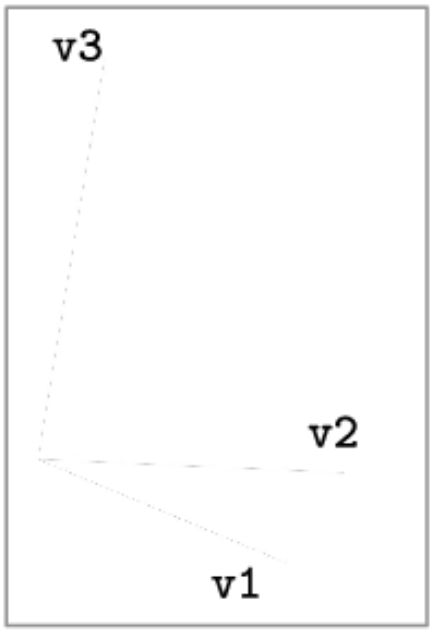
Cross Product

```
import numpy as np

v1 = np.array([1,0,0])
v2 = np.array([0,1,0])

v3 = np.cross(v1, v2)

print(v3)
```

← vecteur **orthogonal** à v1 et v2

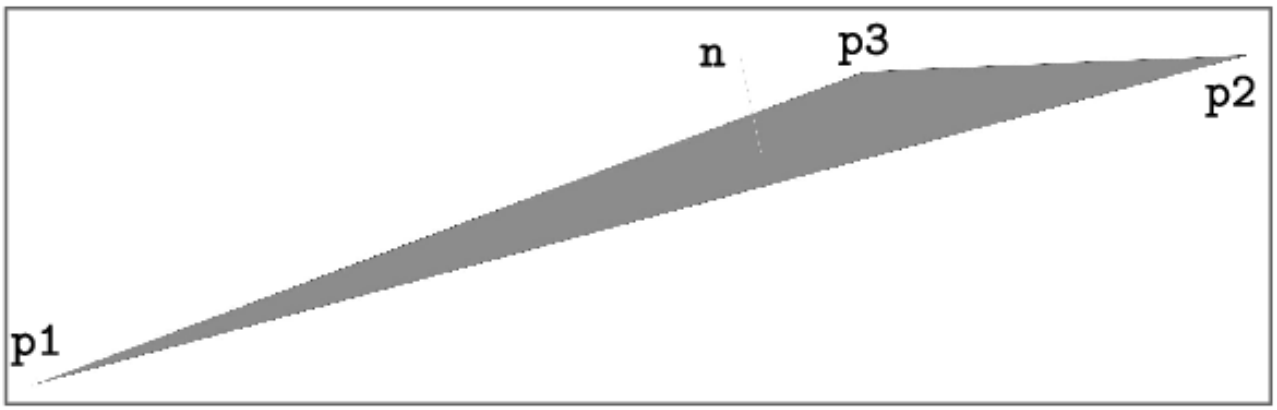
↓

```
[0 0 1]
```



```
p1 = np.array([5.0,2.0,-3.0])
p2 = np.array([-2.0,6.0,6.0])
p3 = np.array([4.0,-3.0,1.0])

v1 = p2 - p1
v2 = p3 - p2
n = np.cross(v1,v2)
n = n / np.linalg.norm(n)
```

vecteur **orthogonal** à triangle (face) `p1-p2-p3`



new idea

Numpy Matrices

```
import numpy as np

A = np.array( [ [1.0, 2.2, 34], [3, 5, 1], [33, 12, -4] ] )

print(A)
print(type(A))
print(A.shape)
```

```
[[ 1.   2.2 34. ]
 [ 3.   5.   1. ]
 [33.  12.  -4. ]]

<class 'numpy.ndarray'>

(3, 3)
```

```
import numpy as np

A = np.array( [ [1.0, 2.2, 34] ] )

print(A)
print(type(A))
print(A.shape)
```

```
[[ 1.   2.2 34. ]]

<class 'numpy.ndarray'>

(1, 3)
```

```
import numpy as np

A = np.array( [ [1.0, 2.2, 34], [3, 5, 1], [33, 12, -4] ] )

print(A[0][0])
print(A[1][2])
print(A[2][2])
```

```
1.0
1.0
-4.0
```

$$\begin{bmatrix} 1.0 & 2.2 & 34 \\ 3 & 5 & 1 \\ 33 & 12 & -4 \end{bmatrix}$$

```
import numpy as np

A = np.array( [ [1.0, 2.2, 34], [3, 5, 1], [33, 12, -4] ] )

print(A[0, 0])
print(A[1, 2])
print(A[2, 2])
```

```
import numpy as np

A = np.array( [ [1.0, 2.2, 34], [3, 5, 1], [33, 12, -4] ] )

print(A)

A[1][1] = -344

print(A)
```

```
[[ 1.   2.2 34. ]
 [ 3.   5.   1. ]
 [33.  12.  -4. ]]

[[   1.    2.2   34. ]
 [   3.  -344.    1. ]
 [  33.   12.   -4. ]]
```

matrix multiplication

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6]])

B = np.array([[7, 8],
              [9, 10],
              [11, 12]])

C = A @ B
print(C)
```

```
[[ 58  64]
 [139 154]]
```

matrix transpose

```
import numpy as np

A = np.array( [ [1.0,2.2,34],
                [3,5,1],
                [33, 12, -4] ] )

C = A.T
print(C)

D = np.transpose(A)
print(D)
```

```
[[ 1.   3.  33. ]
 [ 2.2  5.  12. ]
 [34.   1.  -4. ]]

[[ 1.   3.  33. ]
 [ 2.2  5.  12. ]
 [34.   1.  -4. ]]
```

new idea

Vectors vs Matrices

Careful: a vector vs. a matrix

```
import numpy as np

v1 = np.array([1, 2, 3, 4])
v2 = np.array([[1, 2, 3, 4]])

print(v1.shape)
print(v2.shape)
```

```
(4,)
(1, 4)
```

En Python, `(4)` est interprété comme un simple entier entre parenthèses, donc pour créer un tuple à un seul élément, on ajoute une virgule `(4,)` afin de lever toute ambiguïté.

```
import numpy as np

v1 = np.array([1, 2, 3, 4])
v2 = np.array([[1, 2, 3, 4]])
v3 = np.array([[1], [2], [3], [4]])

print(v1.shape)
print(v2.shape)
print(v3.shape)
```

→

```
(4,)
(1, 4)
(4, 1)
```

vector
1 × 4 matrix
4 × 1 matrix

Careful: a vector vs. a matrix

```
import numpy as np

v1 = np.array([1, 2, 3, 4])
v2 = np.array([[1, 2, 3, 4]])

print(v1.T.shape)
print(v2.T.shape)
```

↓

```
(4,)
(4, 1)
```

```
import numpy as np

v1 = np.array([1, 2, 3, 4])
v2 = np.array([[1, 2, 3, 4]])

print(v1[0])
print(v2[0])
```

↓

```
1
[1 2 3 4]
```

$$\begin{bmatrix}1&2&3&4\\5&6&7&8\end{bmatrix} \cdot \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

Careful: a vector vs. a matrix

```
import numpy as np

v1 = np.array([1, 2, 3, 4])
A = np.array([[1,2,3,4],[5,6,7,8]])
print(A@v1)
```

↓

```
[30 70]
```

```
import numpy as np

v2 = np.array([[1, 2, 3, 4]])
A = np.array([[1,2,3,4],[5,6,7,8]])
print(A@v2)
```

↓

$$\begin{bmatrix}1&2&3&4\\5&6&7&8\end{bmatrix} \cdot [1\ 2\ 3\ 4]$$

```
Traceback (most recent call last):
  File  /home/nabil/t.py , line 5, in <module>
    print(A@v2)
          ~~~~
ValueError: matmul: Input operand 1 has a mismatch in
     its core dimension 0, with gufunc signature (n?,k
     ),(k,m?)->(n?,m?) (size 1 is different from 4)
```

stack: convert a set of vectors into a matrix

```
import numpy as np

vectors = [np.array([1,2,3]),
           np.array([4,5,6]),
           np.array([7,8,9])]

A = np.stack(vectors, axis=0)
B = np.stack(vectors, axis=1)

print(A)
print(B)
```

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

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

new idea

Slicing Matrices

slice each dimension independently: `A[ , ]`

	col 0	col 1	col 2
row 0	1	2	3
row 1	4	5	6
row 2	7	8	9

```
import numpy as np

A = np.array( [ [1,2,3],
                [4,5,6],
                [7,8,9] ] )
```

toutes les rows, toutes les columns

```
print( A[:, :] )
```

↓

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

rows $\{1, 2\}$, columns $\{0, 1\}$

```
print( A[1:3, 0:2] )
```

↓

```
[[4 5]
 [7 8]]
```

rows $\{0, 1, 2\}$, columns $\{1, 2\}$

```
print( A[:, 1:3] )
```

↓

```
[[2 3]
 [5 6]
 [8 9]]
```

slice each dimension independently: `A[ , ]`

	col 0	col 1	col 2
row 0	1	2	3
row 1	4	5	6
row 2	7	8	9

```
import numpy as np

A = np.array( [ [1,2,3],
                [4,5,6],
                [7,8,9] ] )
```

rows $\{2\}$, columns $\{0, 1, 2\}$

```
print( A[2, :] )
```

↓

```
[7 8 9]
```

its a vector!

rows $\{2\}$, columns $\{0, 1, 2\}$

```
print( A[ [2], :] )
```

↓

```
[[7 8 9]]
```

its a matrix with one row!

new idea

Creating Vectors, Matrices

```
import numpy as np
A = np.arange(2, 4, 0.3)
print(A)
```

```
[2.  2.3 2.6 2.9 3.2 3.5 3.8]
```

```
import numpy as np
A = np.arange(5)
print(A)
```

```
[0 1 2 3 4]
```

```
import numpy as np
A = np.linspace(2, 3, 6)
print(A)
```

```
[2.  2.2 2.4 2.6 2.8 3. ]
```

```
import numpy as np

A = np.zeros((2,3))
B = np.ones((3,2))

print(A)
print(B)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

```
import numpy as np

A = np.identity(3)

print(A)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

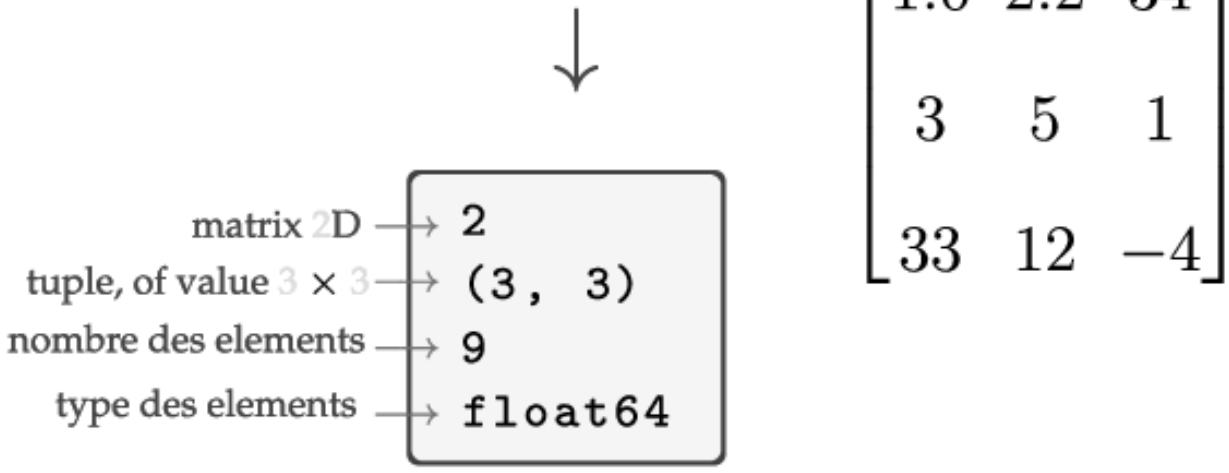
```
import numpy as np
A = np.random.random((3,3))
print(A)
```

```
[[0.32969657 0.33973861 0.59718091]
 [0.41281254 0.54502896 0.61060986]
 [0.96173424 0.69225283 0.38015366]]
```

new idea

Matrices: reshape, resize

```python
import numpy as np

A = np.array( [ [1.0,2.2,34], [3,5,1], [33, 12, -4] ] )

print(A.ndim)
print(A.shape)
print(A.size)
print(A.dtype)
```



$$\begin{bmatrix} 1.0 & 2.2 & 34 \\ 3 & 5 & 1 \\ 33 & 12 & -4 \end{bmatrix}$$

reshape: Returns a new copy of the array with the specified dimensions without changing total number of elements.

```python
import numpy as np

A = np.array( [1, 2, 3, 4, 5, 6] )

B = A.reshape((3,2))
print(B)

C = A.reshape((2,3))
print(C)
```

```
[[1 2]
 [3 4]
 [5 6]]

[[1 2 3]
 [4 5 6]]
```

reshape: Returns a new copy of the array with the specified dimensions without changing total number of elements.

```
import numpy as np

A = np.array( [ [1,2], [3,4] ] )
print(A)

v = A.reshape(4)
print(v)
```

```
[[1 2]
[3 4]]

[1 2 3 4]
```

-1 in reshape automatically infers that dimension's size based on the total number of elements and the other specified dimensions.

```
import numpy as np

A = np.array( [ [1,2], [3,4] ] )
print(A)

v = A.reshape(-1)
print(v)
```

```
[[1 2]
 [3 4]]

[1 2 3 4]
```

```
import numpy as np

A = np.array([[1,2,3],[4,5,6]])
print(A)

v = A.reshape( (3, -1) )
print(v)
```

```
[[1 2 3]
 [4 5 6]]

[[1 2]
 [3 4]
 [5 6]]
```

reshape: taille doit rester la meme

```
import numpy as np

A = np.array( [1, 2, 3, 4, 5, 6] )

D = A.reshape((3,3))
print(D)
```

```
Traceback (most recent call last):
  File  /home/test/tmp/t.py , line 5, in <module>
    D = A.reshape((3,3))
        ^^^^^^^^^^^^^^^^
ValueError: cannot reshape array of size 6 into shape (3,3)
```

```
import numpy as np

A = np.array( [1, 2, 3, 4, 5, 6] )

D = A.reshape((4,-1))
print(D)
```

```
Traceback (most recent call last):
  File  /home/nabil/t.py , line 5, in <module>
    D = A.reshape((4,-1))
ValueError: cannot reshape array of size 6 into shape (4,newaxis)
```

resize: returns a new array with the specified shape, repeating or truncating the original data as needed to fit the new size.

resize does it in two steps:

1. convert into a single list
2. break into rows and columns

```
import numpy as np

A = np.array( [1, 2, 3, 4, 5] )

A.resize((3,3))
print(A)
```

```
[[1 2 3]
 [4 5 0]
 [0 0 0]]
```

```
import numpy as np

A = np.array( [1, 2, 3, 4, 5] )

A.resize((2,2))
print(A)
```

```
[[1 2]
 [3 4]]
```

resize: returns a new array with the specified shape, repeating or truncating the original data as needed to fit the new size.

resize does it in two steps:

1. convert into a single list
2. break into rows and columns

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

A.resize((2,10))
print(A)
```

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

```
import numpy as np

A = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])

A.resize((4,4))
print(A)
```

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

Question: Resultat ?

```
import numpy as np

A = np.arange(10).reshape(2,5)

print(A)
```

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

```
import numpy as np

A = np.array( [2,1,5] )
B = np.array( [56,42,3] )

C = np.outer(A,B)
print(C)
```

```
[[112  84   6]
 [ 56  42   3]
 [280 210  15]]
```

$$C = \begin{bmatrix}2\\1\\5\end{bmatrix} [56 \;\; 42 \; 3] = \begin{bmatrix}2 \cdot 56 & 2 \cdot 42 & 2 \cdot 3\\ 1 \cdot 56 & 1 \cdot 42 & 1 \cdot 3\\ 5 \cdot 56 & 5 \cdot 42 & 5 \cdot 3\end{bmatrix}$$

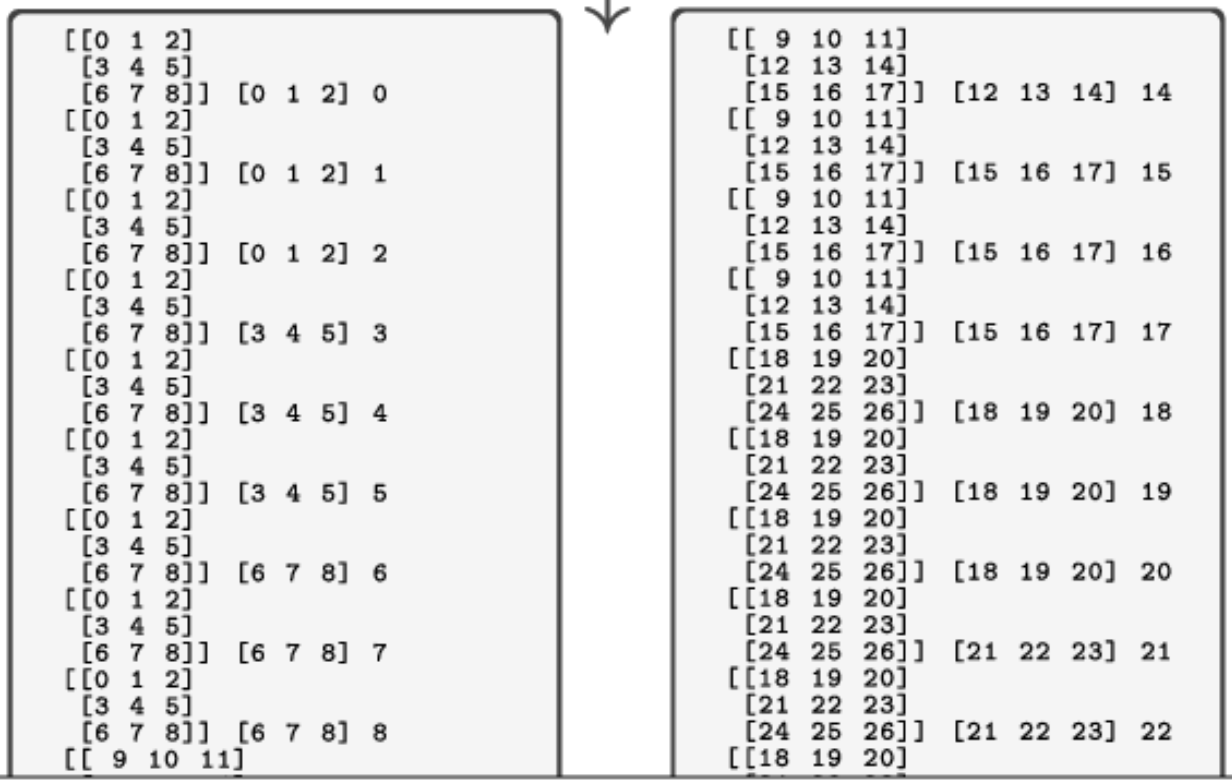
```
import numpy as np

A = np.array( [2,1,5] )
B = np.array( [56,42,3] )

Am = A.reshape(-1, 1)
Bm = B.reshape(1,-1)

D = Am @ Bm
print(D)
```

```
[[112  84   6]
 [ 56  42   3]
 [280 210  15]]
```

```
import numpy as np

A = np.arange(27).reshape((3,3,3))

print(A)
```

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

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]
  [24 25 26]]]
```

```
import numpy as np
A = np.arange(27).reshape((3,3,3))
for a in A:
    for b in a:
        for c in b:
            print(a,b,c)
```



Operation	Code	Shape Rule
Matrix multiplication	`A @ B`	$(m, n)@(n, p) = (m, p)$
Element-wise multiply	`A * B`	Same shape
Transpose	`A.T`	$(m, n) \to (n, m)$
Solve $Ax = b$	`np.linalg.solve(A, b)`	(n, n) and $(n,)$ or (n, m)
Dot product (1D)	`np.dot(a, b)`	$(n,)$ and $(n,) \to$ scalar
Reshape	`A.reshape(2, -1)`	Total elements constant
Flatten to 1D	`A.reshape(-1)`	$(m, n) \to (m \times n,)$
Identity matrix	`np.identity(n)`	(n, n)