

SEMESTRE 1 / COURS 6 - MODULES

ITC MPSI & PCSI – Année 2025-2026



1. Modules
2. Exemples de modules

MODULES

QU'EST-CE QU'UN MODULE ?

- module : fichier python qui contenant des objets informatiques
 - ◇ variables,
 - ◇ fonctions,
 - ◇ classes (programmation orientée objet).

QU'EST-CE QU'UN MODULE ?

- module : fichier python qui contenant des objets informatiques
 - ◇ variables,
 - ◇ fonctions,
 - ◇ classes (programmation orientée objet).
- chaque module regroupe des objets reliés par une même thématique,

QU'EST-CE QU'UN MODULE ?

- module : fichier python qui contenant des objets informatiques
 - ◇ variables,
 - ◇ fonctions,
 - ◇ classes (programmation orientée objet).
- chaque module regroupe des objets reliés par une même thématique,
- nom générique utilisé `module_filename.py`,

QU'EST-CE QU'UN MODULE ?

- module : fichier python qui contenant des objets informatiques
 - ◇ variables,
 - ◇ fonctions,
 - ◇ classes (programmation orientée objet).
- chaque module regroupe des objets reliés par une même thématique,
- nom générique utilisé `module_filename.py`,
- procédé d'importation permettant l'utilisation des objets du module dans d'autres scripts python : mot clé **import**.

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 1

- syntaxe générale

```
from module_filename import objet
```


MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 1

- syntaxe générale

`from module_filename import objet`

- sans importation préalable :

```
>>> cos(pi)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'cos' is not defined. Did you \
↳ mean: 'os'?
```

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 1

- syntaxe générale

`from module_filename import objet`

- sans importation préalable :

```
>>> cos(pi)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'cos' is not defined. Did you \
↳ mean: 'os'?
```

- avec importation préalable :

```
>>> from math import cos,pi
>>> cos(pi)
-1.0
```

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 1

- syntaxe générale

`from module_filename import objet`

- sans importation préalable :

```
>>> cos(pi)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
NameError: name 'cos' is not defined. Did you \
↳ mean: 'os'?
```

- avec importation préalable :

```
>>> from math import cos,pi
>>> cos(pi)
-1.0
```

- inconvénient : objets utilisés pas toujours connus dès le début

- Importation de tous les objets d'un module :

```
from module_filename import *
```

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 2

- Importation de tous les objets d'un module :

```
from module_filename import *
```

exemple :

```
>>> from math import *  
>>> sin(pi/2)  
1.0
```

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 2

- Importation de tous les objets d'un module :

```
from module_filename import *
```

exemple :

```
>>> from math import *  
>>> sin(pi/2)  
1.0
```

- Inconvénient : utilisation de la mémoire et conflit entre fonctions.

MÉTHODE POSSIBLE D'IMPORTATION : MÉTHODE 2

Exemple de conflit :

- importation a

```
>>> from math import *
>>> from turtle import *
>>> radians(90)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: radians() takes 0 positional arguments \
↳ but 1 was given
```

- importation b

```
>>> from turtle import *
>>> from math import *
>>> radians(90)
1.5707963267948966
```

- importation et renommage du module :

```
import module_filename as nom_alias
```


MÉTHODE RECOMMANDÉE D'IMPORTATION AVEC ALIAS

- importation et renommage du module :

```
import module_filename as nom_alias
```

- accès aux objets par la notation pointée :

```
nom_alias.objet
```

MÉTHODE RECOMMANDÉE D'IMPORTATION AVEC ALIAS

- importation et renommage du module :

```
import module_filename as nom_alias
```

- accès aux objets par la notation pointée :

```
nom_alias.objet
```

- Exemple :

```
import math as m
import turtle as t
t.radians()
m.radians(90)
m.sin(m.pi/2)
```

Des informations peuvent être obtenues à l'aide des instructions suivantes :

- `dir(module_filename)` renvoie l'ensemble des objets présents dans un module,

Des informations peuvent être obtenues à l'aide des instructions suivantes :

- `dir(module_filename)` renvoie l'ensemble des objets présents dans un module,
- `help(module_filename)` renvoie des informations sur les fonctions du module,

Des informations peuvent être obtenues à l'aide des instructions suivantes :

- `dir(module_filename)` renvoie l'ensemble des objets présents dans un module,
- `help(module_filename)` renvoie des informations sur les fonctions du module,
- `help(module_filename.objet)` ou `help(nom_alias.objet)` (dans le cas où on a importé et renommé le module) renvoie l'aide relative à cet objet.

Mais pour tout savoir, rien ne vaut le site officiel

<https://docs.python.org/fr!>

INFORMATIONS SUR UN MODULE

Exemple : retour sur le conflit

- ```
>>> import math as m
>>> help(m.radians)
Help on built-in function radians in module math:

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

# INFORMATIONS SUR UN MODULE

## Exemple : retour sur le conflit

- ```
>>> import math as m
>>> help(m.radians)
Help on built-in function radians in module math:
```

```
radians(x, /)
```

```
    Convert angle x from degrees to radians.
```

- ```
>>> import turtle as t
>>> help(t.radians)
Help on function radians in module turtle:
```

```
radians()
```

```
 Set the angle measurement units to radians.
```

```
 No arguments.
```

```
 Example:
```

```
>>> heading()
```

```
90
```

```
>>> radians()
```

```
>>> heading()
```

```
1.5707963267948966
```

## EXEMPLES DE MODULES

---



## QUELQUES MODULES

- `math` – module de fonctions mathématiques usuelles,

## QUELQUES MODULES

- `math` – module de fonctions mathématiques usuelles,
- `cmath` – module autour des nombres complexe ( $a+bj$ ),

## QUELQUES MODULES

- `math` – module de fonctions mathématiques usuelles,
- `cmath` – module autour des nombres complexe ( $a+bj$ ),
- `random` – module de fonctions de hasard et de probabilité,

## QUELQUES MODULES

- `math` – module de fonctions mathématiques usuelles,
- `cmath` – module autour des nombres complexe ( $a+bj$ ),
- `random` – module de fonctions de hasard et de probabilité,
- `numpy` – Module principal de calcul scientifique. Il permet la définition et la manipulation efficace de tableaux. Il fournit des outils pour l'algèbre linéaire, pour l'analyse de Fourier, pour la manipulation de nombres pseudo-aléatoires, ...

## QUELQUES MODULES

- `math` – module de fonctions mathématiques usuelles,
- `cmath` – module autour des nombres complexe ( $a+bj$ ),
- `random` – module de fonctions de hasard et de probabilité,
- `numpy` – Module principal de calcul scientifique. Il permet la définition et la manipulation efficace de tableaux. Il fournit des outils pour l'algèbre linéaire, pour l'analyse de Fourier, pour la manipulation de nombres pseudo-aléatoires, ...
- `matplotlib` et le sous module `matplotlib.pyplot` – module qui permet de créer des graphiques de type courbes, histogrammes, spectres, barres d'erreur, ...

## QUELQUES MODULES

- **math** – module de fonctions mathématiques usuelles,
- **cmath** – module autour des nombres complexe ( $a+bj$ ),
- **random** – module de fonctions de hasard et de probabilité,
- **numpy** – Module principal de calcul scientifique. Il permet la définition et la manipulation efficace de tableaux. Il fournit des outils pour l'algèbre linéaire, pour l'analyse de Fourier, pour la manipulation de nombres pseudo-aléatoires, ...
- **matplotlib** et le sous module **matplotlib.pyplot** – module qui permet de créer des graphiques de type courbes, histogrammes, spectres, barres d'erreur, ...
- **scipy** - module de calcul scientifique qui complète **numpy** et **matplotlib**. Il fournit des outils d'optimisation, d'algèbre linéaire, de statistiques, de traitement du signal, du traitement d'images, de résolution d'équations différentielles.

- importation standardisée :

```
import numpy as np
```

# MODULE NUMPY

- importation standardisée :

```
import numpy as np
```

- fonctions de calcul numérique : statistique, algèbre linéaire, traitement de données :



- importation standardisée :

```
import numpy as np
```

- fonctions de calcul numérique : statistique, algèbre linéaire, traitement de données :

Exemple :

```
>>> import numpy as np
>>> np.sin(np.pi/2)
np.float64(1.0)
>>> np.sqrt(2)
np.float64(1.4142135623730951)
```

# MODULE NUMPY

- importation standardisée :

```
import numpy as np
```

- fonctions de calcul numérique : statistique, algèbre linéaire, traitement de données :

Exemple :

```
>>> import numpy as np
>>> np.sin(np.pi/2)
np.float64(1.0)
>>> np.sqrt(2)
np.float64(1.4142135623730951)
```

- nombreuses fonctions, mais risques de confusions : par exemple, la fonction `randint(a, b)`, où `a` et `b` sont deux entiers, renvoie un entier :
  - ◇ compris dans `[a, b]` pour la fonction du module `random`,
  - ◇ compris dans `[a, b[` pour la fonction du module `np.random`.

- Manipulation de **tableaux** multidimensionnels,

- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.

- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.
- Tableau unidimensionnel : représentation possible en ligne (ou en colonne)  $T = (t_i), i$  entier,

- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.
- Tableau unidimensionnel : représentation possible en ligne (ou en colonne)  $T = (t_i)$ ,  $i$  entier,
- Tableau bidimensionnel : représentation possible en matrice  $T = (t_{i,j})$ ,  $i, j$  entiers,

- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.
- Tableau unidimensionnel : représentation possible en ligne (ou en colonne)  $T = (t_i)$ ,  $i$  entier,
- Tableau bidimensionnel : représentation possible en matrice  $T = (t_{i,j})$ ,  $i, j$  entiers,
- Tableau de dimension quelconque :  $T = (t_{i_1, i_2, \dots})$ ,  $i_1, i_2, \dots$  entiers,

- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.
- Tableau unidimensionnel : représentation possible en ligne (ou en colonne)  $T = (t_i)$ ,  $i$  entier,
- Tableau bidimensionnel : représentation possible en matrice  $T = (t_{i,j})$ ,  $i, j$  entiers,
- Tableau de dimension quelconque :  $T = (t_{i_1, i_2, \dots})$ ,  $i_1, i_2, \dots$  entiers,
- manipulation sur les tableaux régis par des règles précises,



- Manipulation de **tableaux** multidimensionnels,
- Tableau : objet permettant le stockage et la manipulation d'éléments
  - ◇ de nature identiques (tableau d'entiers, tableau de flottants, tableau de booléens, ...)
  - ◇ repérés par un ou plusieurs indices.
- Tableau unidimensionnel : représentation possible en ligne (ou en colonne)  $T = (t_i)$ ,  $i$  entier,
- Tableau bidimensionnel : représentation possible en matrice  $T = (t_{i,j})$ ,  $i, j$  entiers,
- Tableau de dimension quelconque :  $T = (t_{i_1, i_2, \dots})$ ,  $i_1, i_2, \dots$  entiers,
- manipulation sur les tableaux régis par des règles précises,
- syntaxe et rapidité adaptés aux calculs numériques.

La création d'un tableau peut s'effectuer par la fonction `array`.

La création d'un tableau peut s'effectuer par la fonction `array`.

```
>>> import numpy as np
>>> T1 = np.array([1,2,3,4])
>>> type(T1)
<class 'numpy.ndarray'>
>>> T2 = np.array([[1,2,3],[4,5,6]])
>>> T2
array([[1, 2, 3],
 [4, 5, 6]])
```

- $T[i, j]$  : renvoie l'élément d'indice  $(i, j)$  du tableau, situé à la ligne  $(i + 1)$  et à la colonne  $(j + 1)$  (pour tableau 2-D)

- `T[i,j]` : renvoie l'élément d'indice  $(i,j)$  du tableau, situé à la ligne  $(i + 1)$  et à la colonne  $(j + 1)$  (pour tableau 2-D)
- `T.shape` ou `np.shape(T)` : renvoie taille du tableau sous forme d'un tuple

# MANIPULATION DE TABLEAUX

- `T[i,j]` : renvoie l'élément d'indice  $(i,j)$  du tableau, situé à la ligne  $(i + 1)$  et à la colonne  $(j + 1)$  (pour tableau 2-D)
- `T.shape` ou `np.shape(T)` : renvoie taille du tableau sous forme d'un tuple

Exemple :

```
>>> import numpy as np
>>> T2 = np.array([[1,2,3],[4,5,6]])
>>> T2[1,0]
np.int64(4)
>>> T2.shape
(2, 3)
>>> np.shape(T2)
(2, 3)
```

- `T[i,:]` : renvoie les éléments de la ligne  $(i + 1)$  (sous forme d'un tableau)

- $T[i, :]$  : renvoie les éléments de la ligne  $(i + 1)$  (sous forme d'un tableau)
- $T[:, j]$  : renvoie les éléments de la colonne  $(j + 1)$  (sous forme d'un tableau)



- `T[i,:]` : renvoie les éléments de la ligne  $(i + 1)$  (sous forme d'un tableau)
- `T[:,j]` : renvoie les éléments de la colonne  $(j + 1)$  (sous forme d'un tableau)

Exemple :

```
>>> import numpy as np
>>> T2 = np.array([[1,2,3],[4,5,6]])
>>> T2[1,:]
array([4, 5, 6])
>>> T2[:,1]
array([2, 5])
```

- `np.zeros([a, b, c, ...])` : renvoie un tableau rempli de zéros de dimensions  $a, b, c, \dots$

- `np.zeros([a, b, c, ...])` : renvoie un tableau rempli de zéros de dimensions  $a, b, c, \dots$
- `np.ones([a, b, c, ...])` : renvoie un tableau remplis de 1 de dimensions  $a, b, c, \dots$

# MANIPULATION DE TABLEAUX

- `np.zeros([a, b, c, ...])` : renvoie un tableau rempli de zéros de dimensions  $a, b, c, \dots$
- `np.ones([a, b, c, ...])` : renvoie un tableau remplis de 1 de dimensions  $a, b, c, \dots$
- `np.linspace(a, b, n)` : création d'un tableau unidimensionnel de  $n$  valeurs flottantes comprises entre  $a$  et  $b$  (tous deux inclus)

- `np.zeros([a, b, c, ...])` : renvoie un tableau rempli de zéros de dimensions  $a, b, c, \dots$
- `np.ones([a, b, c, ...])` : renvoie un tableau rempli de 1 de dimensions  $a, b, c, \dots$
- `np.linspace(a, b, n)` : création d'un tableau unidimensionnel de  $n$  valeurs flottantes comprises entre  $a$  et  $b$  (tous deux inclus)
- `np.arange(a, b, c)` : création d'un tableau unidimensionnel de valeurs flottantes comprises entre  $a$  (inclu) et  $b$  (exclu) par pas de  $c$

# MANIPULATION DE TABLEAUX

- `np.zeros([a, b, c,...])` : renvoie un tableau rempli de zéros de dimensions  $a, b, c, \dots$
  - `np.ones([a, b, c,...])` : renvoie un tableau remplis de 1 de dimensions  $a, b, c, \dots$
  - `np.linspace(a, b, n)` : création d'un tableau unidimensionnel de  $n$  valeurs flottantes comprises entre  $a$  et  $b$  (tous deux inclus)
  - `np.arange(a, b, c)` : création d'un tableau unidimensionnel de valeurs flottantes comprises entre  $a$  (inclu) et  $b$  (exclu) par pas de  $c$
- Exemple :

```
>>> np.linspace(0,10,11)
array([0., 1., 2., 3., 4., 5., 6., 7., \
↪ 8., 9., 10.])
>>> np.arange(0,11,1)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \
↪ 10])
```

# OPÉRATIONS SUR LES TABLEAUX

Les opérations sont réalisées élément par élément (on parle de vectorisation)

- somme de deux tableaux de même taille :

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

alors que pour deux listes, on obtient :

```
>>> A = [1,2,3]
>>> B = [4,5,6]
>>> A+B
[1, 2, 3, 4, 5, 6]
```

- multiplication par un nombre

```
>>> A = np.array([1,2,3])
>>> 2*A
array([2, 4, 6])
```



- multiplication par un nombre

```
>>> A = np.array([1,2,3])
>>> 2*A
array([2, 4, 6])
```

alors que pour une liste on a :

```
>>> A = [1,2,3]
>>> 2*A
[1, 2, 3, 1, 2, 3]
```

- multiplication par un nombre

```
>>> A = np.array([1,2,3])
>>> 2*A
array([2, 4, 6])
```

alors que pour une liste on a :

```
>>> A = [1,2,3]
>>> 2*A
[1, 2, 3, 1, 2, 3]
```

- élévation d'un tableau à une puissance

```
>>> A = np.array([1,2,3])
>>> A**3
array([1, 8, 27])
```

# OPÉRATIONS SUR LES TABLEAUX

Avec des tableaux de dimension 2 :

```
>>> A = np.array([[1,2],[3,4],[5,6]])
>>> A
array([[1, 2],
 [3, 4],
 [5, 6]])
>>> 2*A # produit de chaque coefficient par un réel
array([[2, 4],
 [6, 8],
 [10, 12]])
>>> A**2 # chaque coefficient au carré
array([[1, 4],
 [9, 16],
 [25, 36]])
```

# OPÉRATIONS SUR LES TABLEAUX

Avec des tableaux de dimension 2 :

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

```
>>> A
```

```
array([[1, 2],
 [3, 4],
 [5, 6]])
```

```
>>> B = np.array([[0,1],[-1,2],[0,2]])
```

```
>>> A+B # somme des tableaux
```

```
array([[1, 3],
 [2, 6],
 [5, 8]])
```

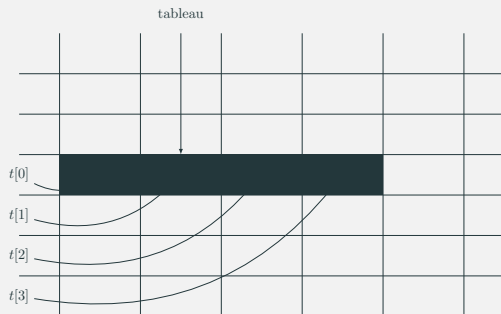
```
>>> A*B # produit coefficient par coefficient
```

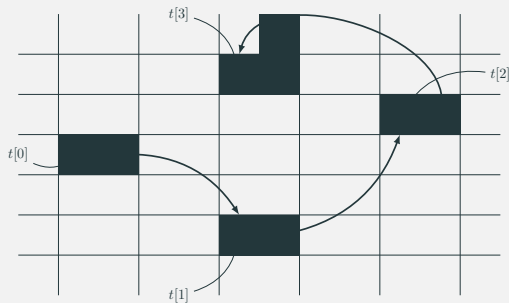
```
array([[0, 2],
 [-3, 8],
 [0, 12]])
```

## FONCTIONS MATHÉMATIQUES ET TABLEAUX

```
>>> import numpy as np
>>> x = np.linspace(0,2,11) # tableau de 11 valeurs \
↪ de 0 à 2
>>> x
array([0. , 0.2, 0.4, 0.6, 0.8, 1. , 1.2, 1.4, 1.6, \
↪ 1.8, 2.])
>>> np.exp(x) # on applique l'exponentielle à chaque \
↪ valeur dans x
array([1. , 1.22140276, 1.4918247 , 1.8221188 \
↪ , 2.22554093,
 2.71828183, 3.32011692, 4.05519997, \
↪ 4.95303242, 6.04964746,
 7.3890561])
```

## Gestion mémoire des tableaux





Gestion mémoire des listes

Normalement, les structures de données liste et tableau ont les propriétés ci-dessous :

|                     | Liste         | Tableau       |
|---------------------|---------------|---------------|
| Structure de donnée | dynamique     | statique      |
| Accès aux données   | coût variable | coût constant |



Normalement, les structures de données liste et tableau ont les propriétés ci-dessous :

|                     | Liste         | Tableau       |
|---------------------|---------------|---------------|
| Structure de donnée | dynamique     | statique      |
| Accès aux données   | coût variable | coût constant |

Attention – listes python :

- possèdent un caractère dynamique,
- temps d'accès à un élément à cout constant.

# LISTES VS TABLEAUX

Normalement, les structures de données liste et tableau ont les propriétés ci-dessous :

|                     | Liste         | Tableau       |
|---------------------|---------------|---------------|
| Structure de donnée | dynamique     | statique      |
| Accès aux données   | coût variable | coût constant |

Attention – listes python :

- possèdent un caractère dynamique,
- temps d'accès à un élément à cout constant.

⇒ Hybrides entre vraies listes et tableaux.

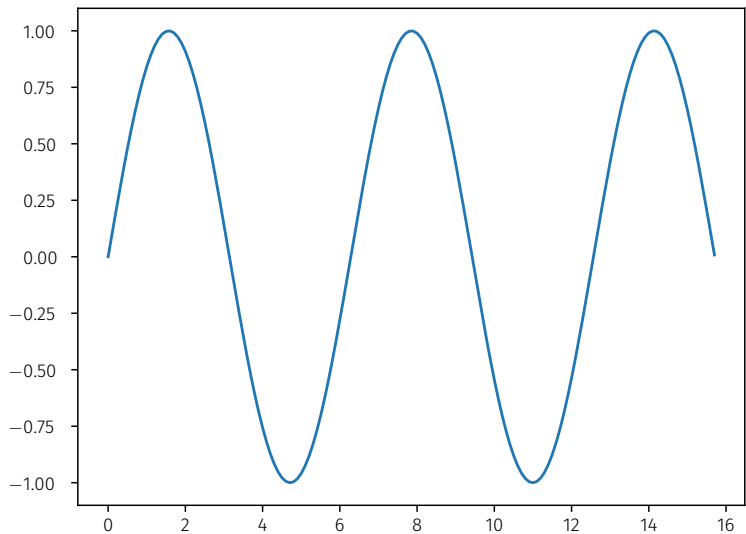
## Module de représentation graphique

Premier exemple :

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0, 5*np.pi, 0.1);
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

## MODULE matplotlib



Deuxième exemple :

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)

y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
```

Deuxième exemple :

```
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'yo-')
plt.title('Deux oscillateurs')
plt.ylabel('Oscillations amorties')

plt.subplot(2, 1, 2)
plt.plot(x2, y2, 'r.-')
plt.xlabel('temps (s)')
plt.ylabel('Oscillations harmoniques')
plt.show()
```

