

Initiation Python

# Organisation du semestre

- 1 CM = 1,5h
- 9 TD/cours (1,5h) = 13,5h
- 4 TP (3h) = 12h
- 27h autant que l'optique par exemple !

	Lundi 15/01/2024	Mardi 16/01/2024	Mercredi 17/01/2024	Judi 18/01/2024	Vendredi 19/01/2024
07h30					
08h00					
08h30		Programmation et algorithmique INFO580_TD1_MISEEPEIPEI TD 2/6 A-64 4CN (44eL)VP- T1487 DECOUT DAMIEN 08h00 - 09h30			Initiation Python TD PYTH201DI_MPC G2 TD 1/9 SD-118 (ex-SD- 113)ISE (2PC-20eL) DECOUT DAMIEN 08h00 - 09h30
09h00					
09h30				INFO580 Informatique INFO580_TP2_BATAIT TP 6/6 E-3 C101 Salle TP Info -12PC- VF interactif (20eL) DECOUT DAMIEN 08h00 - 12h00	Initiation Python TD PYTH201DI_MPC G2 TD 2/9 SC-042 CHR (30p) TB- 3PC-encours DECOUT DAMIEN 09h45 - 11h15
10h00		Initiation Python TD PYTH201DI_MPC G3 TD 1/9 SC-042 CHR (30p) TB- 3PC-encours DECOUT DAMIEN 09h45 - 11h15			
10h30					
10h30					
11h00					
11h30					
12h00					
12h30					
13h00					
13h30		Initiation Python TD PYTH201DI_MPC G4 TD 1/9 SC-042 CHR (30p) TB- 3PC-encours DECOUT DAMIEN 13h15 - 14h45			
14h00					
14h30					
14h30			INFO580 Informatique INFO580_TP2_BATAIT TP 5/6 E-3 C201 Salle TP Info +15 PC- VF interactif (30eL) DECOUT DAMIEN 13h15 - 17h15		INFO580 Informatique INFO580_TP1_BATAIT TP 6/6 E-14 117 HAT (60p) Vidéo-FC TB DECOUT DAMIEN 13h15 - 17h15
15h00					
15h30	Initiation Python TD PYTH201DI_MPC G4 TD 1/9 SC- 036 CHR (2,2L) 20eL PC VP DECOUT DAMIEN 15h00 - 16h30	Initiation Python TD PYTH201DI_MPC G3 TD 2/9 SC-042 CHR (30p) TB- 3PC-encours DECOUT DAMIEN 15h00 - 16h30		REIMOND DUFINO REIMOND 20 8C-244 CHR* (60pL) OLIVIERE CUCILLO 14h00 - 17h30	
16h00					
16h30					
16h30					
17h00	Initiation Python TD PYTH201DI_MPC G4 TD 2/9 SC- 036 CHR (2,2L) 20eL PC VP DECOUT DAMIEN 16h45 - 18h15	Initiation Python TD PYTH201DI_MPC G3 TD 3/9 SC-042 CHR (30p) TB- 3PC-encours DECOUT DAMIEN 16h45 - 18h15		TELSSON DAVID TELPROTENTHANE DUMOND Y 14h00 - 17h30	
17h30					
17h30					
18h00					
18h30					

# Évaluation

- CC pendant les TD/cours et TP (QCM, sur feuille, sur ordi...)
- Absence non justifiée en TP → défaillance
- Appel en CM, TD et TP...

# Contenu

- Voir *moodle* (Gilles Maurin) « Cours d'initiation à la programmation pour les scientifiques non-informaticiens »
- Programmation comme outil pour les sciences comme les math/calculs
- Incontournable en sciences aujourd'hui
- Révolution IA, machine learning, data sciences...

# La science aujourd'hui c'est...

- de plus en plus de données et...
- de plus en plus de calculs

# Exemples

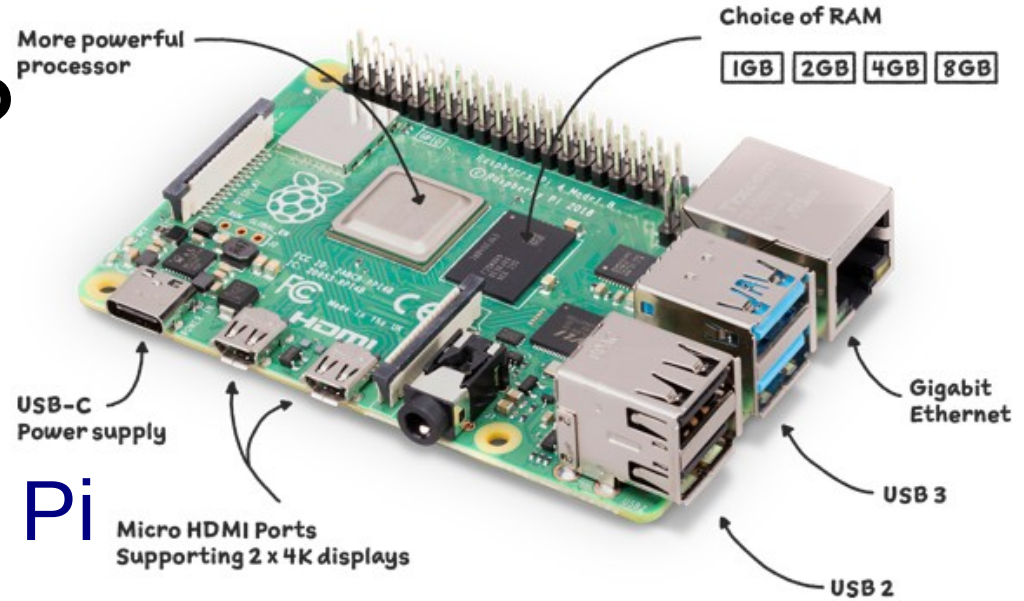
- Fractales (démonstration)
- Météo ([meteoblue](#))
- IA et machine learning en chimie



# L'ordinateur calcule vite, très vite...

- **Supercalculateur**
- Un exaflop = un milliard de milliards d'opérations (en virgule flottante) par seconde !
- Exemples d'utilisation

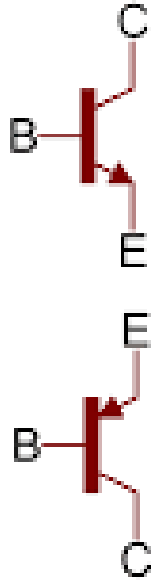
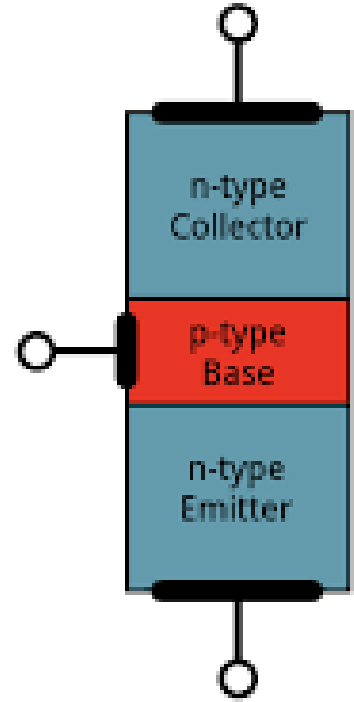
# Rappel ?



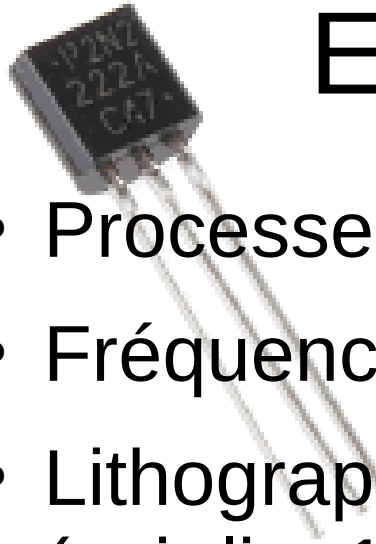
- Exemple du Raspberry Pi
- Périphériques d'entrée et de sortie
- Unité centrale et processeur



# Et mon ordi ?

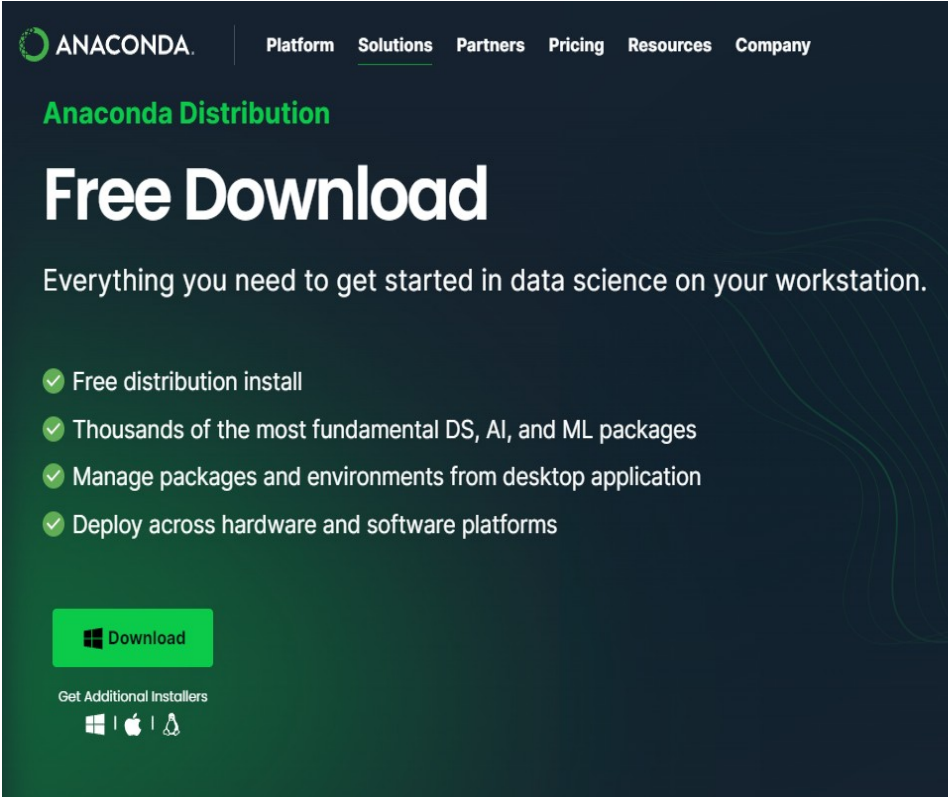


- Processeur
- Fréquence et nombre de coeurs
- Lithographie et nombre de transistors (voir lien1 et lien2 attention aux dates)
- De mon ordi aux supercalculateurs



# Distribution Anaconda

- Installer **Anaconda**
- Lancer Anaconda (d emo)
- Lancer Jupyter-lab (d emo)
- Programmer en Python (d emo)



The screenshot shows the Anaconda website's 'Distribution' page. At the top, the Anaconda logo is on the left, and navigation links for 'Platform', 'Solutions', 'Partners', 'Pricing', 'Resources', and 'Company' are on the right. Below the navigation, the text 'Anaconda Distribution' is displayed in green. The main heading is 'Free Download' in large white font. Underneath, a sub-headline reads 'Everything you need to get started in data science on your workstation.' A list of four benefits follows, each with a green checkmark: 'Free distribution install', 'Thousands of the most fundamental DS, AI, and ML packages', 'Manage packages and environments from desktop application', and 'Deploy across hardware and software platforms'. A prominent green 'Download' button with a white Windows icon is positioned below the list. At the bottom, the text 'Get Additional Installers' is shown above icons for Windows, macOS, and Linux.

# Les cellules Jupyter-lab

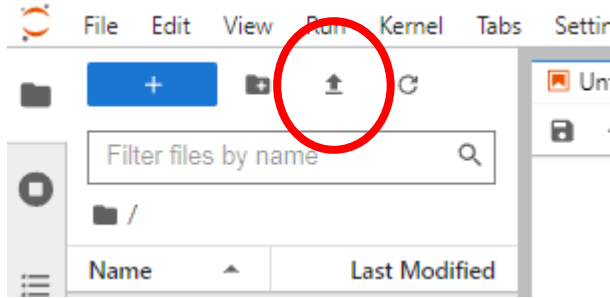
- Du code Python
- Ou du « Markdown »
- Mais aussi Latex et HTML/CSS

# Interface

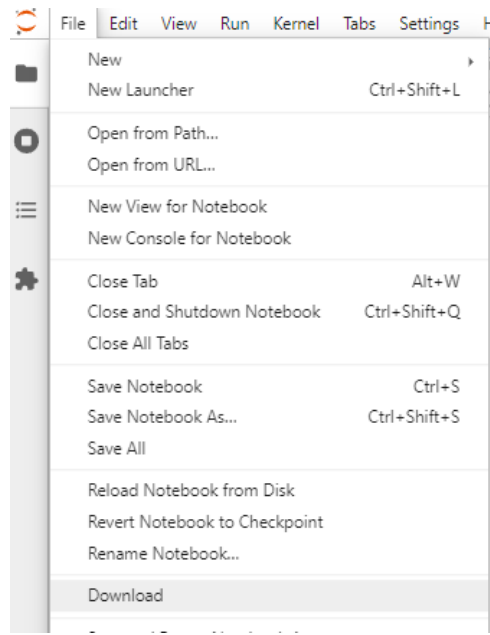
- Python à travers un page web
- Mais en local (le navigateur ne va pas chercher la page web sur un serveur distant)

# Upload and download

Upload « ouvrir »  
(ou glisser déposer)



Download « enregistrer »



# C'est parti !

- On va pouvoir parler à la « machine » pour lui demander de faire des choses
- En utilisant un langage : Python
- En respectant des règles

# shift+Entrée

- Python ne cherche pas à vous comprendre tant que vous ne lui dites pas essaye de me comprendre en validant le code écrit dans la cellule par *shift+Entrée*
- Démo

# Input et output

- Si Python comprend ce que vous avez écrit (input) c'est-à-dire qu'il n'y a pas d'erreur, il affiche éventuellement un résultat (output)
- Démo sans erreur et avec erreur
- Python ne tolère aucune erreur de syntaxe



# Les règles et la pratique

- Connaître la syntaxe est indispensable et ce d'autant plus que Python ne tolère aucune erreur
- Mais ça ne suffit pas (grammaire anglaise/ parler anglais) il faut pratiquer

# Pratiquer

- Lire du « bon code », le comprendre
- Modifier du code
- Écrire du code en commençant par quelques lignes et en augmentant le nombre de lignes progressivement

# Instructions et flux d'exécution

- Pour résoudre un problème, on écrit donc des lignes de code, une suite d'instructions, un programme
- Les instructions sont interprétées (*shift+Entrée*) les unes après les autres dans l'ordre où elles ont été écrites (sauf si le programme contient des instructions conditionnelles ou des instructions répétitives que nous verrons plus loin).
- Attention avec les cellules *Jupyter* ; bonne pratique : revalider tout ce qui suit

# Python tutor

Python 3.6  
[known limitations](#)

```
1 print("debut du code")
2 note = 18
3 print("semestre validé")
→ 4 print("semestre non validé")
→ 5 print("fin du code")
```

[Edit this code](#)

Executed  
Execute

Print output (drag lower right corner to resize)

```
debut du code
semestre validé
semestre non validé
```

Frames

Objects

Global frame

note 18

# Contrôle du flux d'exécution

## Instructions conditionnelles

Python 3.6  
[known limitations](#)

```
1 print("debut du code")
2 note = 18
3 if note>=10:
→ 4     print("semestre validé")
5 else:
6     print("semestre non validé")
→ 7 print("fin du code")
```

[Edit this code](#)

Print output (drag lower right corner to resize)

```
debut du code
semestre validé
```

Frames

Objects

Global frame

note | 18

# Contrôle du flux d'exécution

## Instructions répétitives ou boucles

Python 3.6  
[known limitations](#)

```
1 print("debut du code")
2 note = 18
→ 3 for i in range(5):
→ 4     print("semestre validé")
5 print("semestre non validé")
6 print("fin du code")
```

[Edit this code](#)

executed  
recute

Print output (drag lower right corner to resize)

```
debut du code
semestre validé
semestre validé
semestre validé
```

Frames

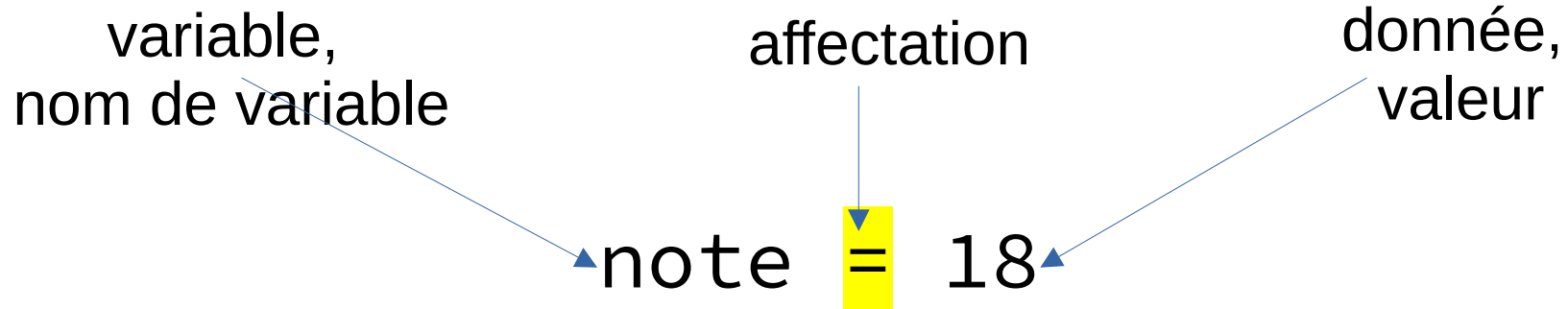
Objects

Global frame

note	18
------	----

i	2
---	---

# Affectation



L'**instruction d'affectation** permet de stocker une **valeur** (dans la mémoire) et de pouvoir la retrouver grâce à un nom de **variable**

# Variable

- Nom choisi par vous, court mais explicite qui devrait évoquer ce que la variable contient pour plus de lisibilité du code
- Mélange de chiffres et de lettres et doit commencer par une lettre
- Différence entre minuscule et majuscule
- Pas d'accent, pas de caractère spéciaux (sauf « \_ »)
- **mots réservés** Python interdits (if et for par exemple)



# Variable examples

`note = 18`

`Note = 18`

`x = 18`

`note_au_premier_CC = 18`

`noteAuPremierCC = 18`

`noteCC1 = 18`

# Réaffectation

- Une variable est associée à un emplacement dans la mémoire qui ne peut contenir qu'une « seule » donnée
- Une autre affectation concernant la même variable écrase donc la donnée précédente

# Réaffectation exemples

```
note = 18
```

```
note = 10
```

```
print(note)
```

```
a = 1
```

```
b = 2
```

```
a = b
```

```
b = a
```

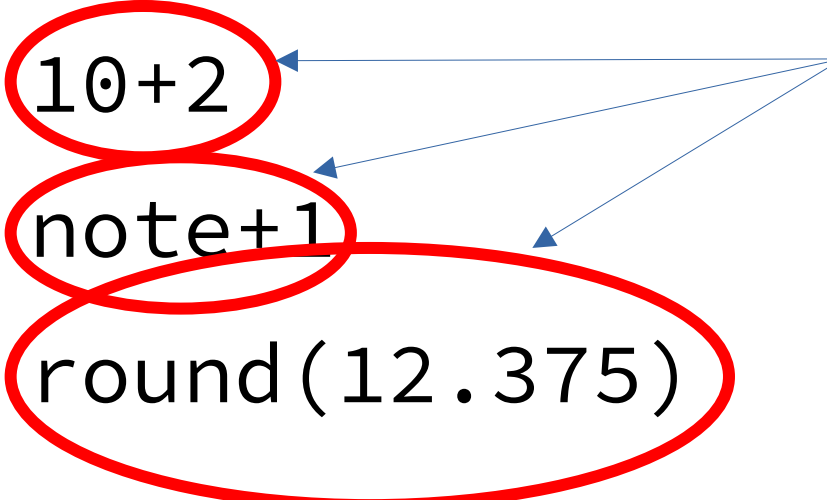
```
print(a,b)
```

# Affectation d'une expression

- Si le terme de droite de l'affectation est une **expression** et non une valeur, Python évalue d'abord l'expression et affecte ensuite le résultat à la variable
- Une expression ne peut pas être stockée dans une variable, seulement son résultat

# Affectation d'une expression exemples

```
note = 10+2
note = note+1
note = round(12.375)
```



Évaluation de  
l'**expression** de  
droite puis  
affectation

# Les fonctions natives

- Voir [doc python](#)
- On a déjà utilisé `print` pour afficher la valeur stockée dans une variable
- On a déjà utilisé `round` qui renvoie l'arrondi du nombre réel passé en paramètre

# Nom fonction et paramètres

nom de la fonction

paramètre(s)

print(note)

Print(3.14)

print("hello world")

print("votre note :", note)

# Composition de fonctions

```
note = 13.175
```

```
print(round(note))
```

Python commence par récupérer la valeur stockée dans `note`, puis la « passe » en paramètre de la fonction `round` qui renvoie la valeur arrondie « passée » en paramètre de la fonction `print` qui l'affiche



# Créer ses propres fonctions

📁 + ✂ 📄 ▶ ■ ↻ ⏩ Code ▾

```
[1]: G = 6.67428e-11 #SI  
m = 70 #kg  
MT = 5.972e24 #kg  
RT = 6371e3 #m  
h = 0 #m
```

```
[2]: force = G*m*MT/(RT+h)**2
```

```
[3]: force
```

```
[3]: 687.3960799998207
```

```
[4]: print(force/9.81)
```

```
70.07095616715807
```

# Créer ses propres fonctions (2)

```
[1]: G = 6.67428e-11 #SI  
     m = 70          #kg  
     MT = 5.972e24  #kg  
     RT = 6371e3    #m  
     h = 0          #m
```

```
[2]: force = G*m*MT/(RT+h)**2
```

```
[3]: force
```

```
[3]: 687.3960799998207
```

```
[4]: print(force/9.81)  
70.07095616715807
```

```
•[1]: G = 6.67428e-11 #SI  
      m = 70          #kg  
      MT = 5.972e24  #kg  
      RT = 6371e3    #m  
      h = 4809       #m
```

```
[2]: force = G*m*MT/(RT+h)**2
```

```
[3]: force
```

```
[3]: 687.3960799998207
```

```
[4]: print(force/9.81)  
70.07095616715807
```

# Créer ses propres fonctions (3)

📁 + ✂ 📄 📄 ▶ ■ 🔄 ▶▶ Code ▾

```
[1]: def force(m,h):  
      G = 6.67428e-11 #SI  
      MT = 5.972e24 #kg  
      RT = 6371e3 #m  
      res = G*m*MT/(RT+h)**2  
      return res
```

```
[2]: force(70,0)
```

```
[2]: 687.3960799998207
```

```
[3]: force(70,4809)
```

```
[3]: 686.3595241433683
```

# Créer ses propres fonctions (4)

```
[6]: masse = 70
alti = 0
print("une personne de",masse,"kg à une altitude de",alti,"m subit une force de",round(force(m,h)), "N")
```

une personne de 70 kg à une altitude de 0 m subit une force de 687 N

- La fonction `Force` peut maintenant être utilisée comme n'importe quelle autre fonction de Python
- Vous avez en quelque sorte « appris » à Python comment calculer la force subie par une masse  $m$  à une altitude  $h$  !
- Et vous pouvez même utiliser cette fonction pour écrire d'autres fonctions !

# Modules et bibliothèques

- Il existe une multitude de fonctions écrites par *la communauté Python* que vous pouvez ré-utiliser dans vos programmes
- **Bibliothèque standard**
- Librairies incontournables *numpy, matplotlib...*
- <https://pypi.org/>