

# **SCRIPTING Python M1**

# Bash vs python



# Bash vs python



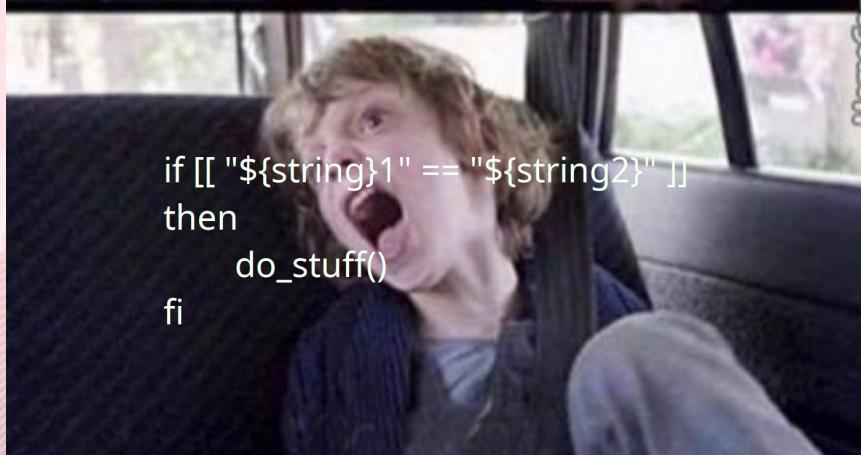
# Bash vs python



Interpréteur de commandes  
Axé sur les commandes linux  
Syntaxe non intuitive sur la prog  
Optimal pour les scripts simples  
Aucune installation  
OS mac et linux

Vrai langage de programmation  
Programmation orientée objets  
Simple à écrire et à comprendre  
Langage polyvalent  
Nécessite d'installer Python  
Multi plateforme (win, mac, linux)  
Accès à de nombreuses bibliothèques

# Bash vs python



# Bash vs python

```
1 nombre_lancers=$1
2 somme_des=0
3 for ((i = 0 ; i < $nombre_lancers ; i++)); do
4     de1=$((RANDOM % 20 + 1))
5     de2=$((RANDOM % 20 + 1))
6     echo "Lancer $i : $de1 et $de2"
7     if [[ $de1 -eq $de2 ]]; then
8         echo "Vous êtes mort !"
9         exit 0
10    fi
11    somme_des=$((somme_des + de1 + de2))
12 done
13
14 echo "Voici le score cumule obtenu : $somme_des !"
```

```
1 from random import randint
2 nombre_lancers=10
3 somme_des=0
4 for i in range(0, nombre_lancers):
5     de1=randint(1,20)
6     de2=randint(1,20)
7     print(f'Lancer {i} : {de1} et {de2}')
8     if de1 == de2:
9         print("Vous êtes mort !")
10        break
11    somme_des=somme_des + de1 + de2
12
13 print(f'Voici le score cumule obtenu : {somme_des} !')
```

# Python

- Structures de données simples et complexes
- Fonctions
- Modules
- Programmation orientée objets

# Comment utiliser Python?

The screenshot shows the Visual Studio Code interface with the following details:

- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help. The "hello\_world.py - python - Visual Studio Code" tab is open.
- Sidebar (Left):** EXPLORER, PYTHON (with "hello\_world.py" selected), SEARCH, SYMBOLS, FILES, CLIPBOARD.
- Editor Area:** Shows the Python code: 

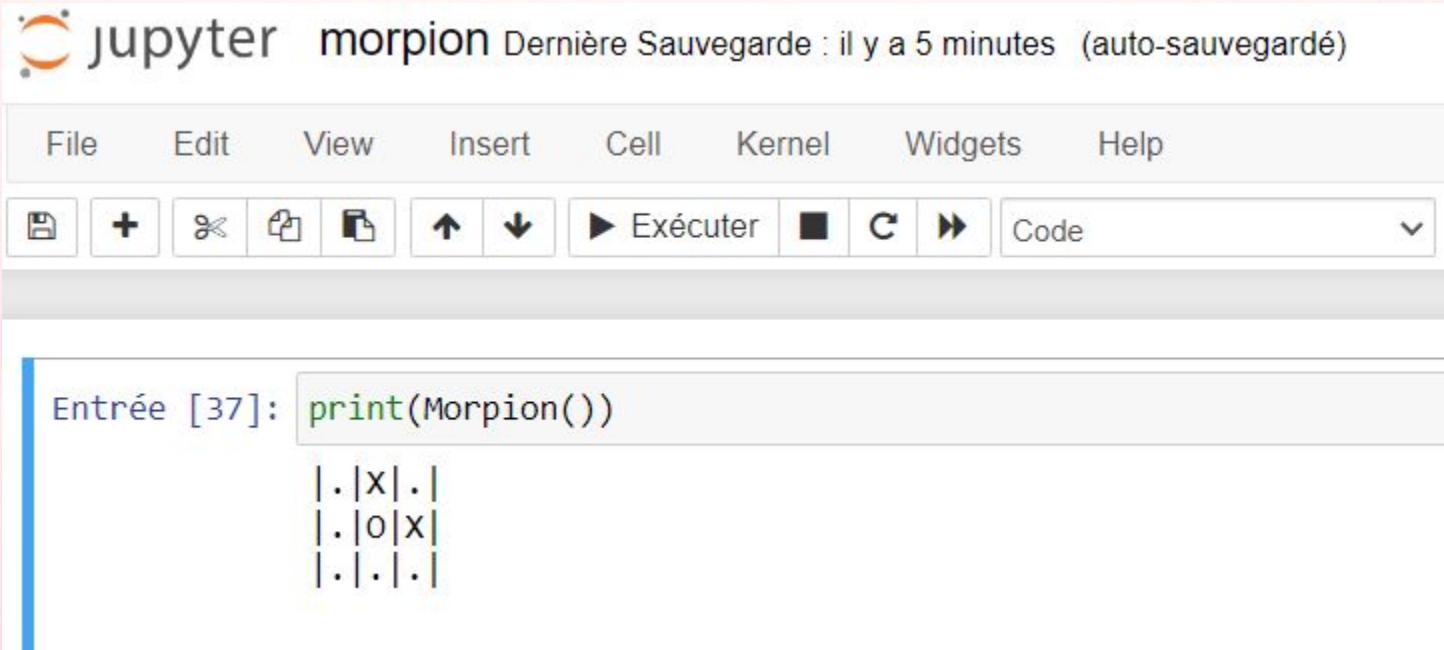
```
1 print("Hello world!")
```
- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined).
- Terminal Output:** Displays the command and its result:
  - PS C:\Users\leosu\OneDrive\Documents\scripting\python> python .\hello\_world.py
  - Hello world!
  - PS C:\Users\leosu\OneDrive\Documents\scripting\python> [redacted]

# Comment utiliser Python?



ANACONDA®

# Comment utiliser Python?



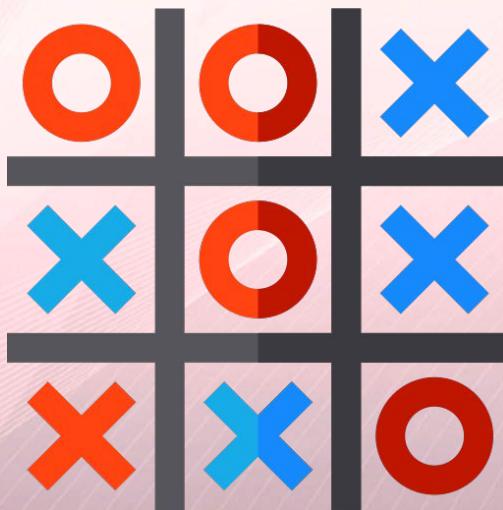
The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter morpion Dernière Sauvegarde : il y a 5 minutes (auto-sauvegardé)
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Toolbar:** Includes icons for file operations (Save, New, Open, Save All, Undo, Redo), cell execution (Exécuter), and kernel management (Kernel, Cell Kernel, Cell Kernel).
- Code Cell:** Entrée [37]: `print(Morpion())`
- Output:** The output of the code cell is a 3x3 grid representing a game board:

```
..|x|..|  
.|.|x|  
..|.|..|
```

# **GR0S** Projet de la séance

Coder un programme Python qui permet de jouer au morpion, puis créer une intelligence artificielle capable de vous battre à ce jeu.



# Point retour sur les variables

# Les variables, c'est quoi?

Fondation de tout langage de programmation

*Eléments qui associent un nom à une valeur,  
qui sont stockés dans la mémoire du système  
programmé*

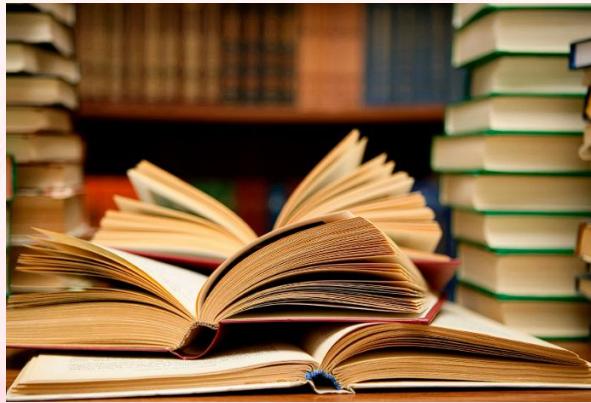
# Métaphore de la bibliothèque



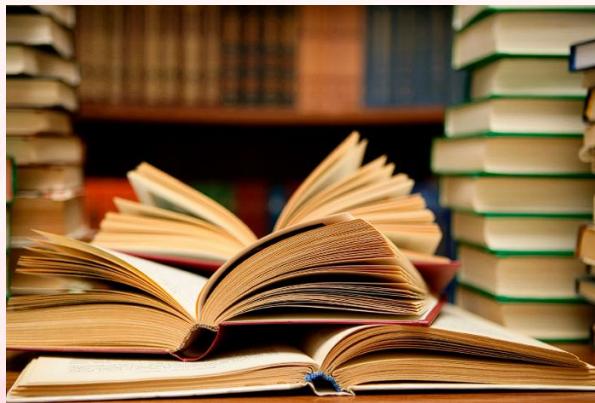
# Métaphore de la bibliothèque



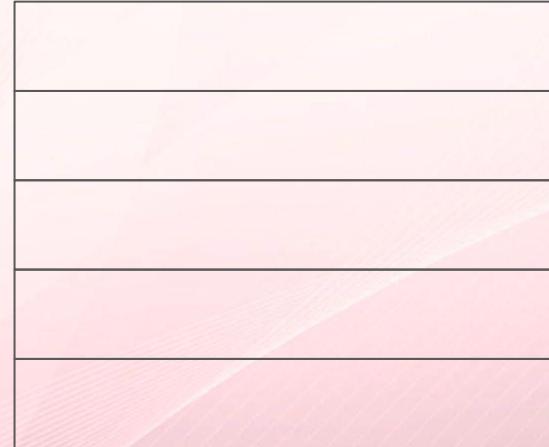
# Métaphore de la bibliothèque



# Métaphore de la bibliothèque



# Métaphore de la bibliothèque

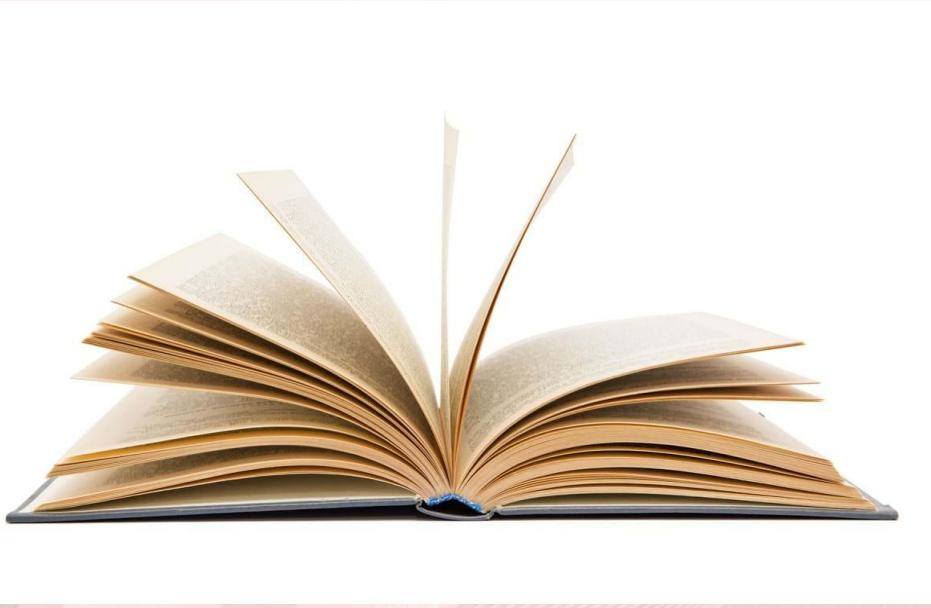


# Métaphore de la bibliothèque

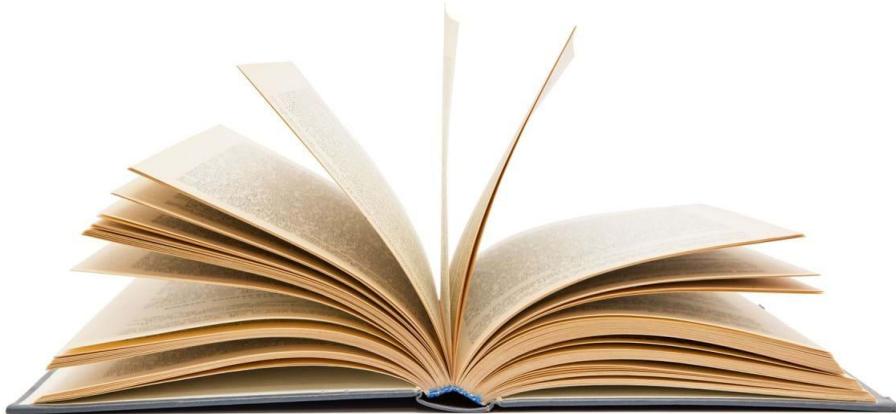


x = 1
y = "coucou"
Z = True

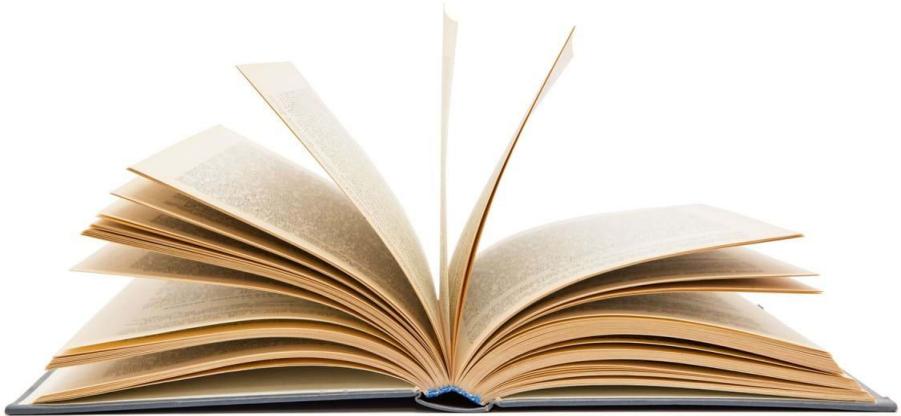




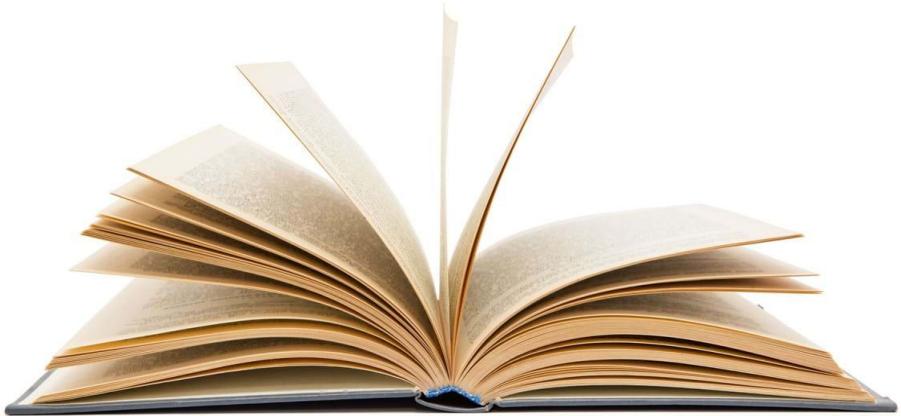
?



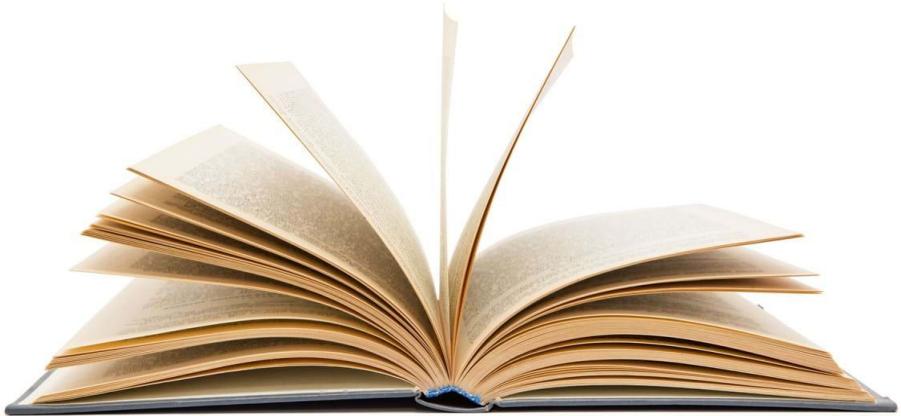
- Titre



- Titre
- Genre



- Titre
- Genre
- Contenu



- Titre
- Genre
- Contenu
- Poids

**X** = **1**

Variable

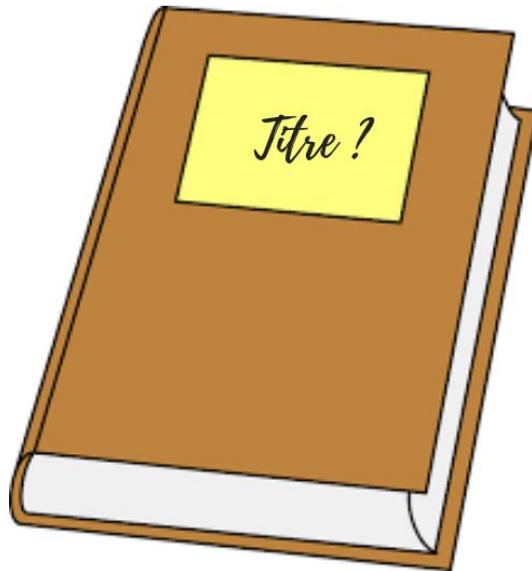
- Titre
- Genre
- Contenu
- Poids

# Choisir le titre d'une variable



Le titre idéal pour mon  
oeuvre ?

# Choisir le titre d'une variable



Le titre idéal pour mon  
oeuvre ?

- Un nom porteur de sens
- En minuscules
- Tout attaché
- Espaces transformés en “\_”

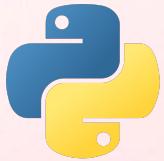
# Choisir le genre d'une variable

roman historique	conte
roman d'aventure	légende
roman fantastique	poésie
roman fantastique épique	BD
roman policier	documentaire animalier
roman noir	documentaire historique
roman socio-réaliste	documentaire scientifique
biographie	documentaire autre sujet

# Choisir le genre d'une variable

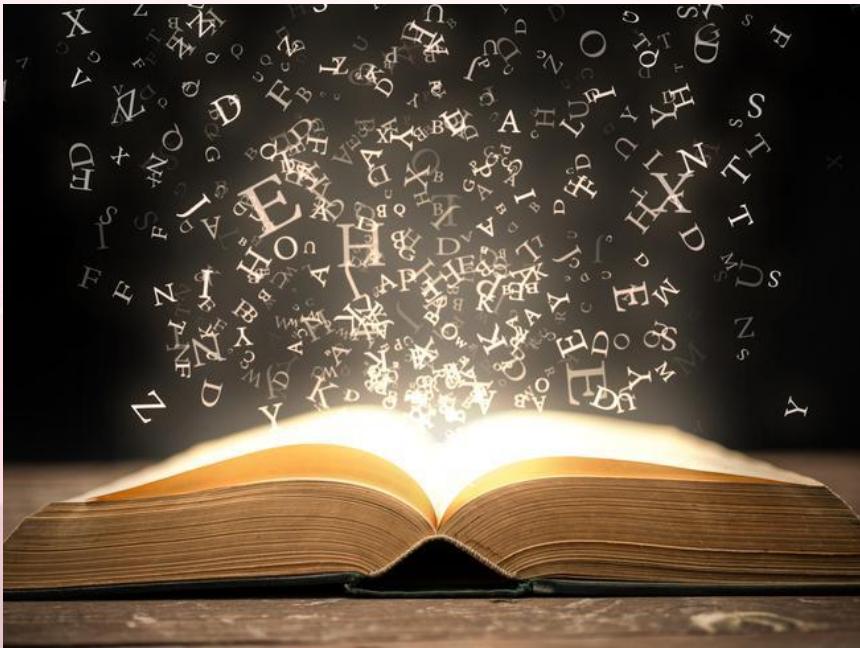
roman historique	conte
roman d'aventure	légende
roman fantastique	poésie
roman fantastique épique	BD
roman policiier	documentaire animalier
roman noir	documentaire historique
roman socio-réaliste	documentaire scientifique
biographie	documentaire autre sujet

## Types simples

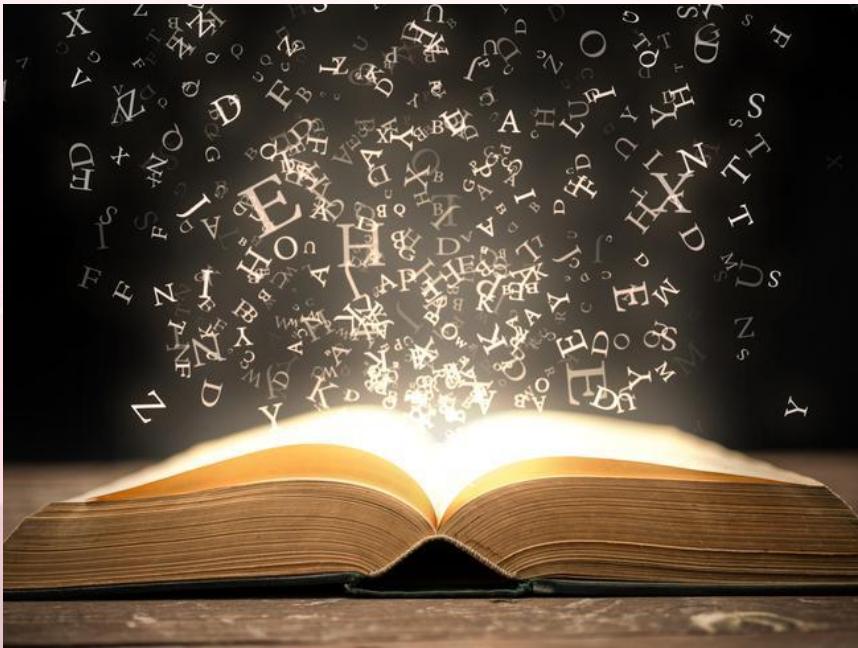


- Nombre entier
- Nombre à virgule
- Chaîne de caractères
- Vrai ou faux

# Ecrire le contenu d'une variable

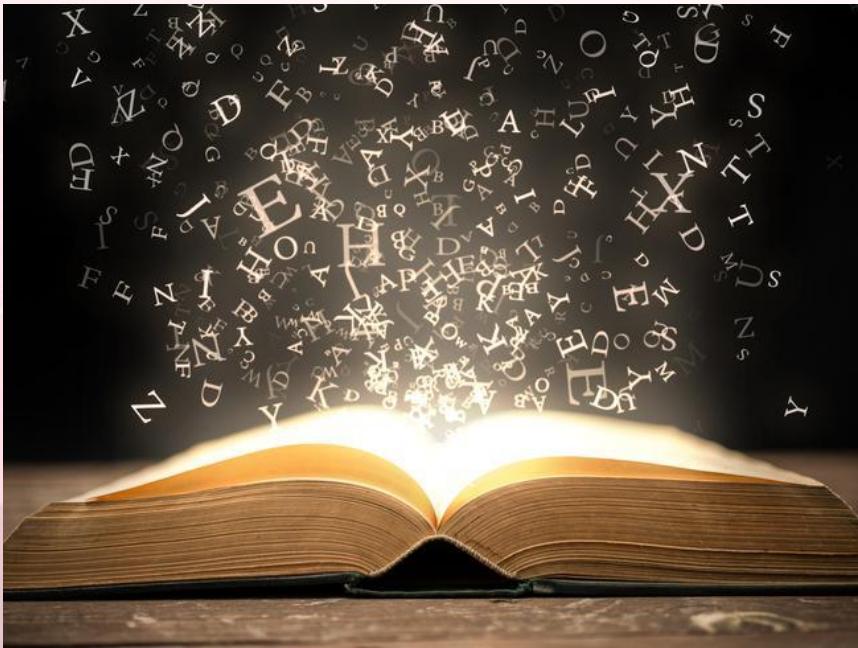


# Ecrire le contenu d'une variable



- Nombre entier
- Nombre à virgule
- Chaîne de caractères
- Vrai ou faux

# Ecrire le contenu d'une variable



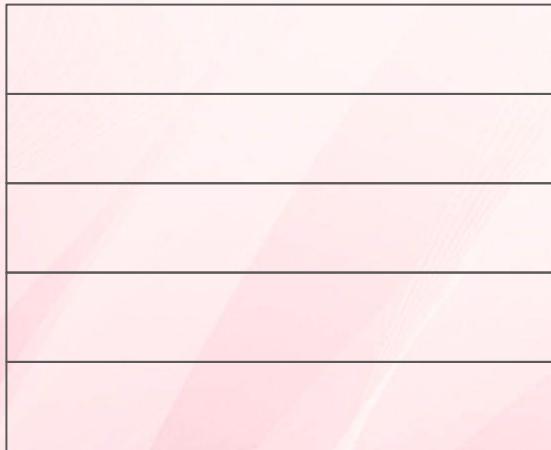
`solution_univers = 42`

`solution_france = 49.3`

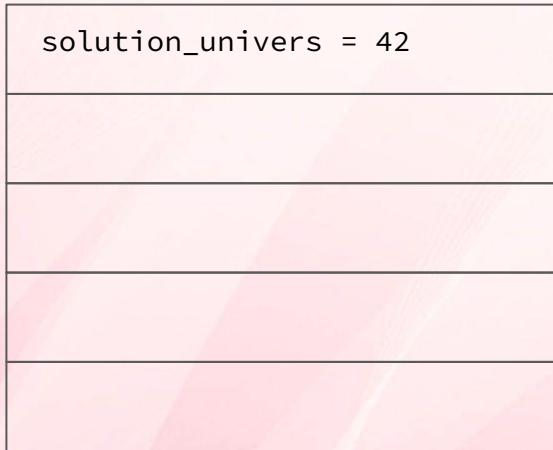
`solution_vie = "bonheur"`

`belle_journee = True`

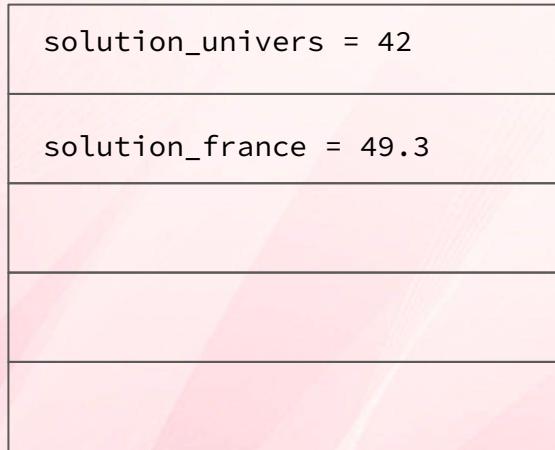
# Fonctionnement de la mémoire



# Fonctionnement de la mémoire



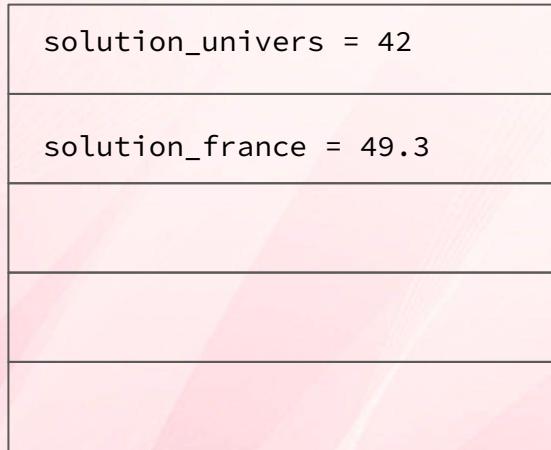
# Fonctionnement de la mémoire



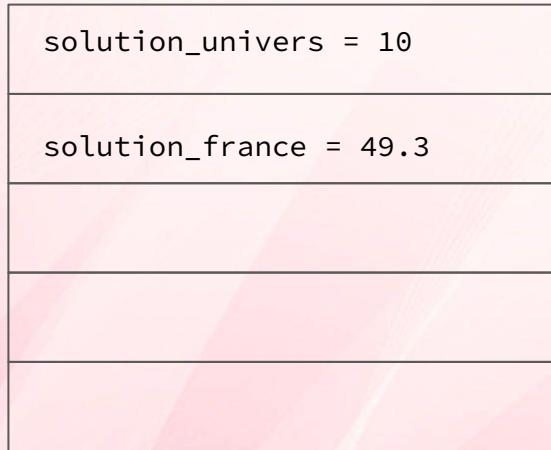
solution\_univers = 42

solution\_france = 49.3

# Fonctionnement de la mémoire



# Fonctionnement de la mémoire



`solution_univers = 42`

`solution_france = 49.3`

`solution_univers = 10`

# Utiliser la valeur d'une variable

```
solution_univers = 42  
solution_france = 49.3
```

solution\_univers = 42

solution\_france = 49.3

distance\_marathon = solution\_univers

# Utiliser la valeur d'une variable

solution_univers = 42
solution_france = 49.3
distance_marathon = 42

solution\_univers = 42

solution\_france = 49.3

distance\_marathon = solution\_univers

# Utiliser la valeur d'une variable

C'est l'histoire de 3 Héros légendaires Gragas, Ornn et Olaf réputés dans le monde entier pour leurs talents de buveurs , mais sans un sou, qui rentrent dans une taverne.

Un écriteau à l'entrée marque "si vous parvenez à finir le bidon de biere en moins de 42 minutes, vous ne payez pas l'addition !

Gragas s'exclame : "Ce bidon, je me l'enfile en une heure, pas une minute de plus" en tapotant son propre bidon

Ornn, fatigué par le voyage répond : "Je ne pourrai en consommer que le tiers de ce que tu prendras, Gragas"

Enfin, olaf blessé par la quête qu'ils ont mené le jour même rétorque : "Et moi seulement la moitié de ce que tu avaleras Ornn.

Peuvent-ils réussir le challenge proposé par la taverne ou n'en sont ils pas capables?

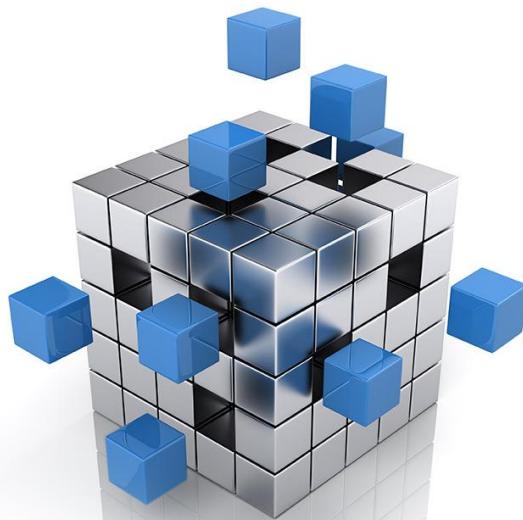
Exprimer le problème en programmation et créer une condition pour tester le résultat



# Structures de données complexes

# La programmation

## Un outil pour modéliser le monde



# La programmation

## Un outil pour modéliser le monde

- int
- float
- booléen
- chaînes de caractères

# La programmation

# Un outil pour modéliser le monde



# La programmation

## Un outil pour modéliser le monde

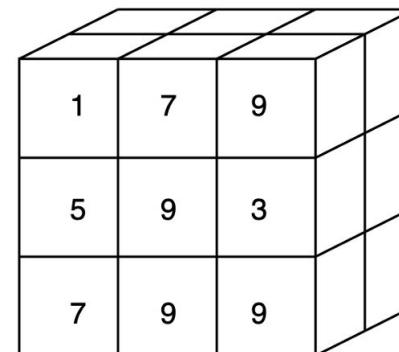
1D Array

3	2
---	---

2D Array

1	0	1
3	4	1

3D Array



# Les structures de données complèxes

Permet de stocker de multiples données dans une seule variable

var1 =

65	118	70	18	64				
15	54	48	51	17				64
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16
57	35	2	75	94	94	13		16

# **La structure la plus puissante**

## **La liste**

Les listes permettent de stocker une série d'éléments

Les éléments d'une liste sont :

- Ordonnés
- Mutables
- Permettent les valeurs doublons

# **La structure la plus puissante**

## **La liste**

```
liste = [1,2,4]
```

**Liste composée de trois éléments, trois nombres entiers (int)**

# **La structure la plus puissante**

## **La liste**

**liste = [1,2,4]**

**Liste composée de trois éléments, trois nombres entiers (int)**

**Le premier élément de la liste est 1**

# **La structure la plus puissante**

## **La liste**

**liste = [1,2,4]**

**Liste composée de trois éléments, trois nombres entiers (int)**

**Le premier élément de la liste est 1**

**Le second élément de la liste est 2**

# La structure la plus puissante

## La liste

```
liste = [1,2,4]
```

Liste composée de trois éléments, trois nombres entiers (int)

Le premier élément de la liste est 1

Le second élément de la liste est 2

Le troisième élément de la liste est 4

# La structure la plus puissante

## La liste

```
liste = [1,2,4]
```

On peut accéder aux éléments de la liste en se servant d'un indice

Le premier élément de la liste est 1

Le second élément de la liste est 2

Le troisième élément de la liste est 4

# La structure la plus puissante

## La liste

liste = [1, 2, 4]

On peut accéder aux éléments de la liste en se servant d'un indice

Le premier élément de la liste est 1

Le second élément de la liste est 2

Le troisième élément de la liste est 4

# La structure la plus puissante

## La liste

liste = [1, 2, 4]

On peut accéder aux éléments de la liste en se servant d'un indice

Le premier élément de la liste est 1    liste[0]

Le second élément de la liste est 2    liste[1]

Le troisième élément de la liste est 4    liste[2]

# La structure la plus puissante

## La liste

liste = [1,2,4]

Ajouter des éléments dans une liste

```
Entrée [4]: liste = [1,2,4]
              liste.append(2)
              print(liste)
```

```
[1, 2, 4, 2]
```

# La structure la plus puissante

## La liste

liste = [1,2,4]

**Modifier une valeur dans une liste**

Entrée [6]:

```
liste = [1,2,4]
liste[1] = 3
print(liste)
```

[1, 3, 4]

# La structure la plus puissante

## La liste

liste = [1,2,4]

### Boucler sur une liste

```
Entrée [2]: liste = [1,2,4]
for el in liste:
    print(el)
```

1  
2  
4

# La structure la plus puissante

## La liste

Créer un script qui définit une liste qui représente une grille de jeu de morpion.

Ensuite, parcourir la grille à l'aide d'une boucle et afficher le plateau avec la fonction print()



# La chaîne de caractères, soeur de la liste

chaîne = “salut”

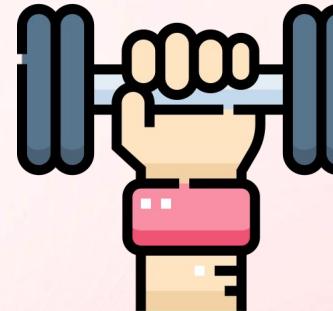
On peut accéder aux éléments de la liste en se servant d'un indice

Le premier élément de la chaîne est s liste[0]

Le second élément de la chaîne est a liste[1]

Le troisième élément de la chaîne est l liste[2]

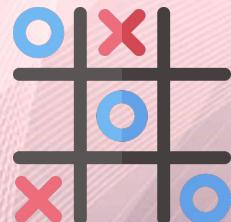
# La chaîne de caractères



Reprendre la liste grille\_morpion définie à l'exercice précédent et créer une chaîne de caractères qui représente cette grille.

La chaîne de caractères sera de taille 9, un vide sera représenté par un “.”, une croix par un “X” et un rond par une “0”.

Exemple :



“OX..O.X.O”

# **La structure la plus rapide le dictionnaire**

Les dictionnaires permettent de stocker des couples clé - valeur

Dans un dictionnaire chaque clé est unique et est associée à une valeur

# Le dictionnaire

```
dico = {"cle" : valeur1, "cle2": valeur2}
```

On accède aux valeurs d'un dictionnaire via leur clé

La clé "cle" a comme valeur valeur1

La clé "cle2" a comme valeur valeur2

```
dico["cle"]  
dico["cle2"]
```

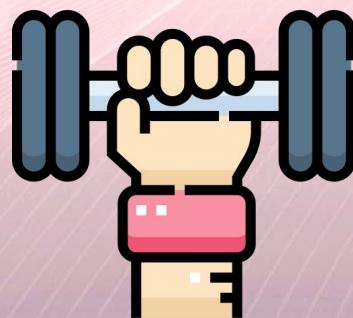
# Le dictionnaire

```
Entrée [16]: bulletin={"Léo": 14, "Marc": 16, "Axel": 15.5}
              for key in bulletin:
                  print(f"{key} : {bulletin[key]}")
```

```
Léo : 14
Marc : 16
Axel : 15.5
```

# Exercice

Parcourir un bulletin de notes et récupérer le nom de l'élève qui a minoré le contrôle et le nom de celui qui l'a majoré et les afficher



# Les mal aimés : Le set et le tuple

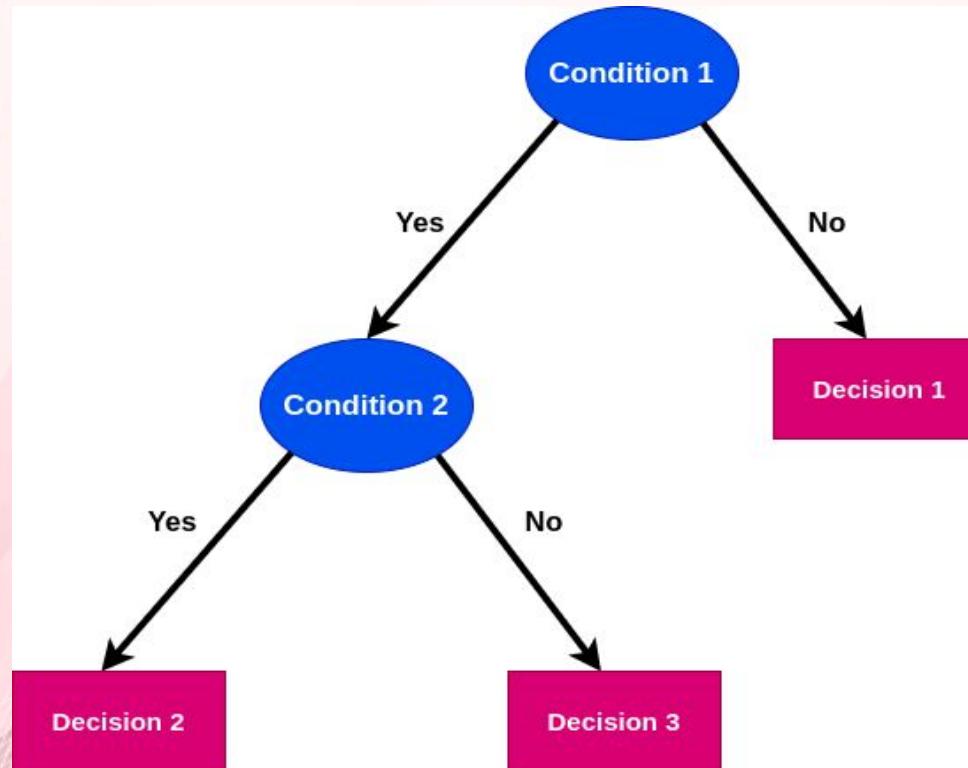
```
ensemble_math = {1, 2, 3} # set
```

```
tuple = (1,2,3) # tuple
```

	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

# Point retour sur les conditions

# Arbre de conditions



# Arbre de conditions

```
if condition1:  
    if condition2:  
        decision="decision2"  
    else:  
        decision="decision3"  
else:  
    decision="decision1"
```

# Arbre de conditions

```
if soleil :  
    if vampire:  
        etat="je meurs"  
    else:  
        etat="je bronze"  
else:  
    etat="je reste chez moi"
```

# Arbre de conditions

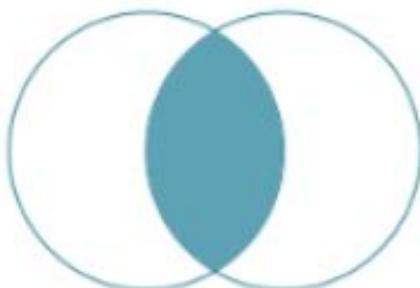
```
if temps_dehors=="soleil" :  
    if ma_race=="vampire":  
        etat="je meurs"  
    else:  
        etat="je bronze"  
else:  
    etat="je reste chez moi"
```

# **Algèbre de boole**

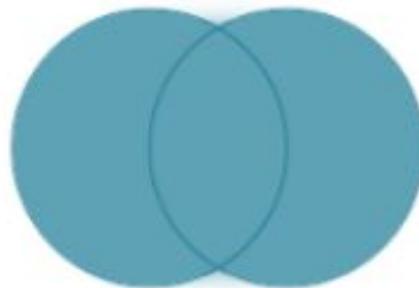
Pour créer des conditions plus complexes

**AND    OR    NOT**

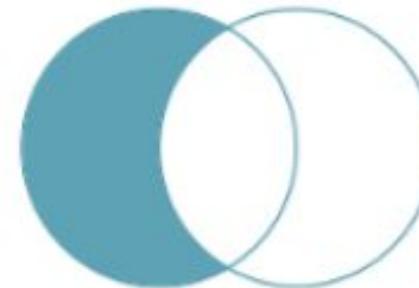
# Algèbre de boole



**ET**  
(AND)



**OU**  
(OR)



**NON**  
(NOT)

# **Algèbre de boole**

Pour créer des conditions plus complèxes

`bool1 or bool2`

# **Algèbre de boole**

Pour créer des conditions plus complèxes

`bool1 and bool2`

# **Algèbre de boole**

Pour créer des conditions plus complèxes

not bool1

# Algèbre de boole AND

bool 1

bool2

résultat

FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

# Algèbre de boole OR

bool 1      bool2      résultat

bool 1	bool2	résultat
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

# Algèbre de boole NOT

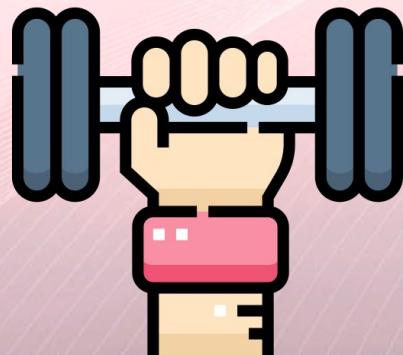
bool 1

résultat

TRUE	FALSE
FALSE	TRUE

# Exercice conditions

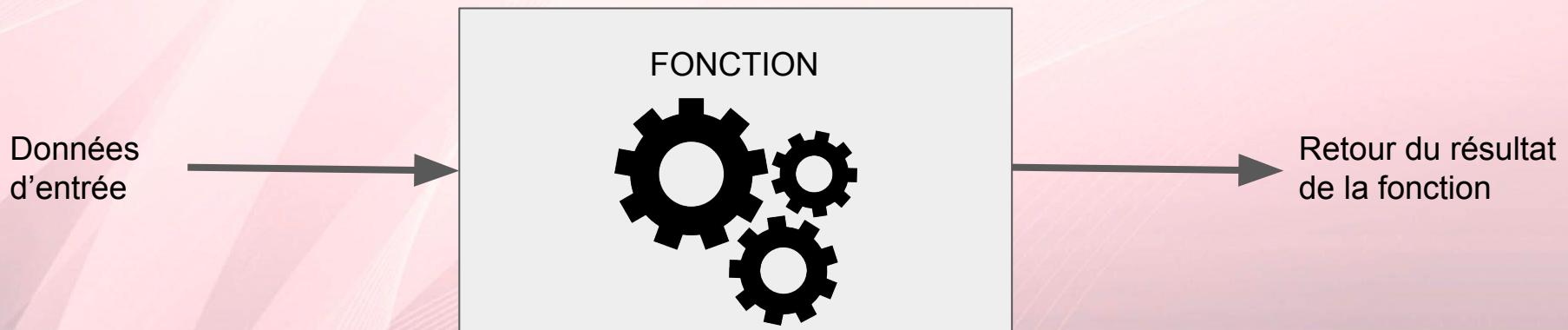
Ecrire un code qui vérifie si une grille de morpion est gagnante



# Les fonctions

# Les fonctions

Les fonctions sont vraiment la quintessence de la programmation, car elles permettent d'encapsuler des petits univers de logiques dans un seul mot, ce qui rend l'outil extrêmement puissant.



# Les fonctions

Les fonctions sont vraiment la quintessence de la programmation, car elles permettent d'encapsuler des petits univers de logiques dans un seul mot, ce qui rend l'outil extrêmement puissant.

```
1 def sum(a,b):  
2     return a + b  
3  
4 resultat1=sum(1,3)  
5 resultat2=sum(2,6)
```

# Les fonctions

Mot clé pour définir une fonction

Nom de la fonction

Paramètres à passer pour utiliser la fonction

```
1 def sum(a,b):  
2     return a + b  
3  
4 resultat1=sum(1,3)  
5 resultat2=sum(2,6)
```

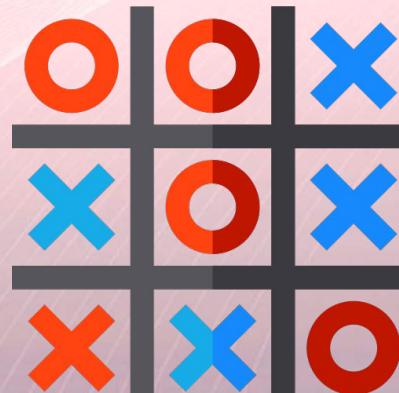
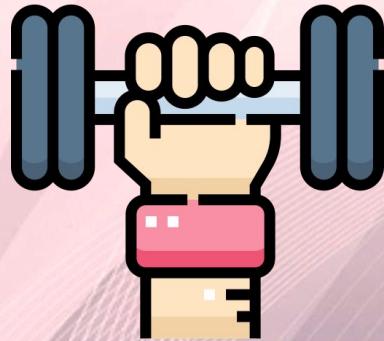
Valeur de retour de la fonction



# EXERCICE

Créer une fonction **position\_win** qui reprend le code que vous avez écrit qui teste si une position d'un plateau est gagnante.

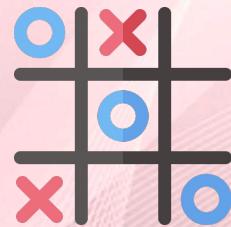
Elle prendra un argument grille\_morpion et retournera true si la position est gagnante, et false dans le cas contraire.



# EXERCICE

Créer une fonction **get\_state** qui renvoie une représentation du plateau sous forme de chaîne de caractères

Elle prendra un argument grille\_morpion et retournera la représentation en chaîne de caractère du plateau



“OX..O.X.O”



# EXERCICE



Créer une fonction **coup\_valide** qui prend en paramètre la grille de morpion, un numéro de ligne et de colonne, et qui retourne true si le coup est valide (aucune croix ni rond n'est déjà placée sur la position, et le numéro de ligne et de colonne est correct), false sinon

Créer une fonction **jouer\_coup** qui prend en paramètre grille\_morpion, numero\_ligne, numero\_colonne et forme (“X” ou “O”) et qui place la forme à la case indiquée



# EXERCICE

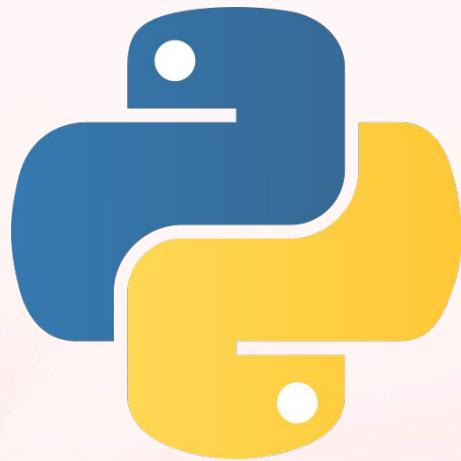
Créer une fonction **play\_random\_move** qui prend en paramètre grille\_morpion et forme (“X” ou “O”) et qui place la forme à une case aléatoire valide sur le plateau



# Les fonctions natives en Python

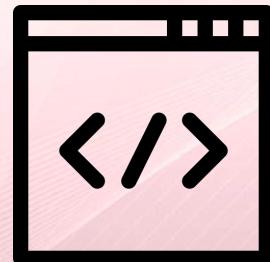
Built-in Functions			
<b>A</b> abs() aiter() all() any() anext() ascii()	<b>E</b> enumerate() eval() exec()	<b>L</b> len() list() locals()	<b>R</b> range() repr() reversed() round()
<b>B</b> bin() bool() breakpoint() bytearray() bytes()	<b>F</b> filter() float() format() frozenset()	<b>M</b> map() max() memoryview() min()	<b>S</b> set() setattr() slice() sorted() staticmethod() str() sum() super()
<b>C</b> callable() chr() classmethod() compile() complex()	<b>G</b> getattr() globals()	<b>N</b> next()	<b>T</b> tuple() type()
<b>D</b> delattr() dict() dir() divmod()	<b>H</b> hasattr() hash() help() hex()	<b>O</b> object() oct() open() ord()	<b>V</b> vars()
	<b>I</b> id() input() int() isinstance() issubclass() iter()	<b>P</b> pow() print() property()	<b>Z</b> zip()
			<b>_</b> <b>__import__()</b>





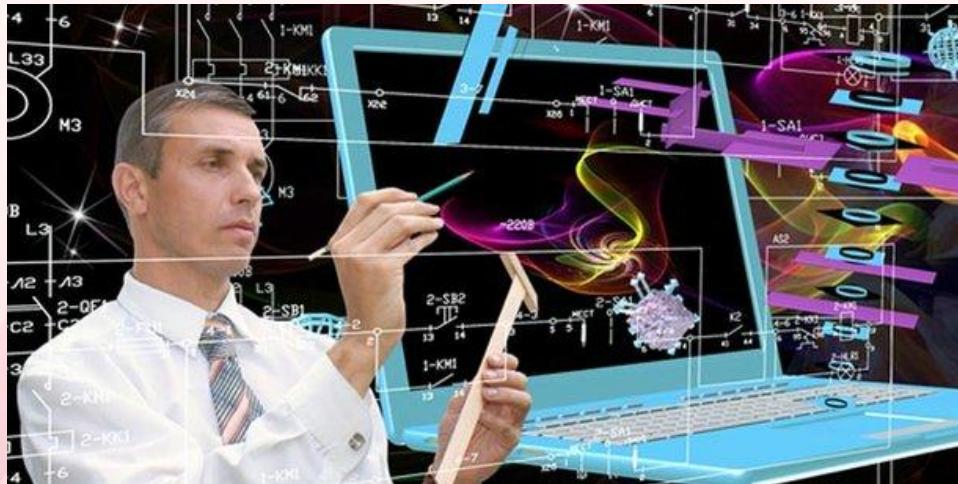
# PROGRAMMATION ORIENTÉE OBJET

# **QU'EST-CE QU'UN BON PROGRAMME?**



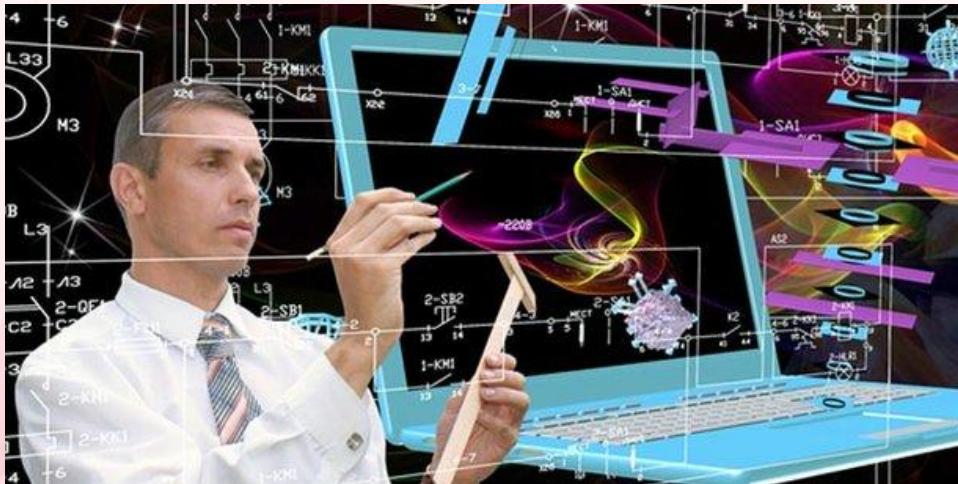
# ENJEU DE LA MODÉLISATION

Enjeu de parvenir à modéliser les problèmes du monde réel “correctement”.



# ENJEU DE LA MODÉLISATION

Enjeu de parvenir à modéliser les problèmes du monde réel “correctement”.



Une bonne modélisation est celle qui répond à des besoins de performance, de simplicité et de flexibilité.

# QU'EST-CE QU'UN BON PROGRAMME?

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

# QU'EST-CE QU'UN BON PROGRAMME?

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## *Simplicité*

- Code clair
- Code maintenable
- Code testable

# QU'EST-CE QU'UN BON PROGRAMME?

</>

## ***Performance***

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## ***Simplicité***

- Code clair
- Code maintenable
- Code testable

## ***Flexibilité***

- Code extensible
- Changements aisés
- Code modulaire

# QU'EST-CE QU'UN BON PROGRAMME?

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## *Simplicité*

- Code clair
- Code maintenable
- Code testable

## *Flexibilité*

- Code extensible
- Changements aisés
- Code modulaire

**Types employés jusqu'à maintenant :**

```
nombre = 5 # INT
flottant = 5.2 # FLOAT
vrai = True # BOOL
chaine = "bonjour" # CHAINE
```

# QU'EST-CE QU'UN BON PROGRAMME?

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## *Simplicité*

- Code clair
- Code maintenable
- Code testable

## *Flexibilité*

- Code extensible
- Changements aisés
- Code modulaire

**Types employés jusqu'à maintenant :**

```
liste = [1,2,3] # LISTE
tup = (1,2,3) # TUPLE
dico = {"clé": "valeur"} # DICT
ensemble = {1,2,3} # SET
```

# QU'EST-CE QU'UN BON PROGRAMME?

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## *Simplicité*

- Code clair
- Code maintenable
- Code testable

## *Flexibilité*

- Code extensible
- Changements aisés
- Code modulaire



# QU'EST-CE QU'UN BON PROGRAMME?

## ***Performance***

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## ***Simplicité***

- Code clair
- Code maintenable
- Code testable

## ***Flexibilité***

- Code extensible
- Changements aisés
- Code modulaire



# QU'EST-CE QU'UN BON PROGRAMME?

</>

## *Performance*

- Rapidité d'exécution
- Mémoire optimisée
- Précision du résultat

## *Simplicité*

- Code clair
- Code maintenable
- Code testable

## *Flexibilité*

- Code extensible
- Changements aisés
- Code modulaire



# RENDRE SON CODE PLUS INTUITIF

</>

Regardez autour de vous !

</>

# RENDRER SON CODE PLUS INTUITIF



# RENDRER SON CODE PLUS INTUITIF



</>

# RENDRER SON CODE PLUS INTUITIF



# LE MONDE DES OBJETS



# UNE INFINITÉ DE POSSIBILITÉS

</>



A LinkedIn profile card for Julia Cames. It features a circular profile picture of a woman with long brown hair, set against a background of a mountain range under a cloudy sky. Below the picture is her name, "Julia Cames", followed by a "2e" badge. A short bio describes her as the Head of Marketing at HubSpot, mentioning expertise in digital growth, brand operations, user experience, marketing, demand management, and team management for B2B & B2C. It also notes her location in Paris, Île-de-France, France, and provides a link to her LinkedIn profile. At the bottom of the card are three buttons: "Se connecter" (Connect), "Message" (Message), and "Plus".



# **MODÉLISER DES OBJETS**



# COMMENT DÉFINIR UN OBJET?



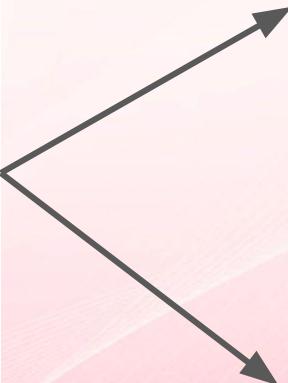
# COMMENT DÉFINIR UN OBJET?



Attributs

Fonctions

# COMMENT DÉFINIR UN OBJET?



Caractéristiques  
inhérentes de l'objet

Fonctionnalités de  
l'objet

# COMMENT DÉFINIR UN OBJET?



- Taille
  - Poids
  - Couleur
  - Intensité
- 
- Allumer
  - Eteindre
  - Plier

# COMMENT DÉFINIR UN OBJET?



- Taille
- Poids
- Couleur
- Intensité
- Matière
- longueur\_depliee
- longueur\_repliee
- circonference
- aire\_bouton
- perimetre\_bouton
- type\_ampoule
- nombre\_vis...

# COMMENT DÉFINIR UN OBJET?



- Taille
- Poids
- Couleur
- Intensité
- Matière
- longueur\_depliee
- longueur\_repliee
- circonference
- aire\_bouton
- perimetre\_bouton
- type\_ampoule
- nombre\_vis...



# MODÉLISATION DES OBJETS



Lampe
Attributs
Méthodes

# MODÉLISATION DES OBJETS



Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

Méthodes

# MODÉLISATION DES OBJETS



Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer() -> None  
eteindre() -> None

# MODÉLISATION DES OBJETS



Lampe

taille : float  
poids : float  
couleur : str  
ampoule : Ampoule

allumer() -> None  
eteindre() -> None

# MODÉLISATION DES OBJETS



Ampoule

puissance: float  
duree\_vie: float  
couleur\_temp : str  
allume: bool

# **EXERCICE DE MODÉLISATION**



**Modéliser les attributs et les méthodes d'une souris d'ordinateur !**

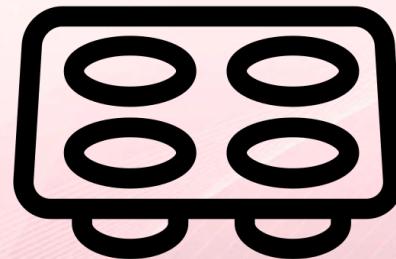


# EXERCICE DE MODÉLISATION



Souris	
poids: float	filaire: bool
couleur: str	largeur: float
nbr_boutons: int	longueur: float
marque: str	hauteur: float
nom_modele: str	
clic_droit()	
clic_gauche()	
clic_molette()	
deplacer curseur(direction)	
scroll(haut:bool)	

# **LA CLASSE : LE MOULE DES OBJETS**





# LA CLASSE : LE MOULE DES OBJETS

Pour fabriquer des objets, on va créer un moule. En python, ce moule est ce qu'on appelle une classe.

# LA CLASSE : LE MOULE DES OBJETS



Pour fabriquer des objets, on va créer un moule. En python, ce moule est ce qu'on appelle une classe.





# LA CLASSE : LE MOULE DES OBJETS

Pour fabriquer des objets, on va créer un moule. En python, ce moule est ce qu'on appelle une classe.





# LES CLASSES EN PYTHON

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

La classe, un moule

```
: class Lampe:  
    # définition des attributs  
  
    # définition des méthodes
```



# LES CLASSES EN PYTHON

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes
```



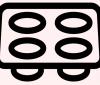
# LES CLASSES EN PYTHON

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

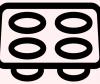
```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```



# LES CLASSES EN PYTHON

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```



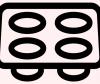


# INSTANCIER DES OBJETS

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```



```
l1 = Lampe()  
l2 = Lampe()  
l3 = Lampe()
```



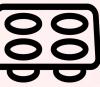
# INSTANCIER DES OBJETS

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```



```
l1 = Lampe()  
l2 = Lampe()  
l3 = Lampe()
```

Pour instancier notre objet,  
on écrit le **nom de notre classe** suivi de **parenthèses** !  
Cela aura pour effet d'appeler  
la méthode spéciale du  
**constructeur** !



# INSTANCIER DES OBJETS

```
l1 = Lampe()  
l2 = Lampe()  
l3 = Lampe()
```



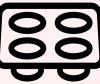
lampe1



lampe2



lampe3



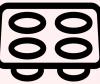
# INSTANCIER DES OBJETS

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, intensite, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.intensite = intensite  
        self.is_on = is_on  
    # définition des méthodes  
    def allumer(self):  
        self.is_on = True  
    def eteindre(self):  
        self.is_on = False
```



C'est dans le **constructeur `__init__`** que nous allons définir les **attributs** de notre classe.

Ainsi, lorsque nous **instancions** notre objet, nous pouvons lui passer en paramètres des valeurs spécifiques pour notre objet !

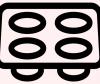


# INSTANCIER DES OBJETS

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, intensite, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.intensite = intensite  
        self.is_on = is_on  
    # définition des méthodes  
    def allumer(self):  
        self.is_on = True  
    def eteindre(self):  
        self.is_on = False
```

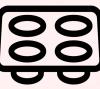


```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)  
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)  
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```



# INSTANCIER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```



# INSTANCIER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```



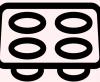
lampe1



lampe2



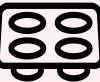
lampe3



# INSTANCIER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```

Chaque instance est un objet indépendant des autres qui a ses propres attributs !



# MANIPULER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```

Accéder aux attributs de  
l'instance

lampe1.couleur

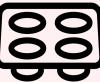
'noir'

lampe2.couleur

'rose'

lampe3.couleur

'blanc'



# MANIPULER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```

Accéder aux attributs de  
l'instance

```
lampe1.couleur
```

```
'noir'
```

```
lampe2.couleur
```

```
'rose'
```

```
lampe3.couleur
```

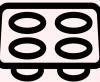
```
'blanc'
```

Utiliser des méthodes de  
l'instance

```
lampe1.eteindre()
```

```
lampe2.eteindre()
```

```
lampe3.allumer()
```



# MANIPULER DES OBJETS

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```

Accéder aux attributs de l'instance

```
lampe1.couleur
```

```
'noir'
```

```
lampe2.couleur
```

```
'rose'
```

```
lampe3.couleur
```

```
'blanc'
```

Utiliser des méthodes de l'instance

```
lampe1.eteindre()
```

```
lampe2.eteindre()
```

```
lampe3.allumer()
```

Manipulations sur l'instance

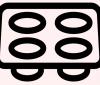
```
lampe1.is_on
```

```
False
```

```
lampe1.allumer()
```

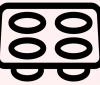
```
lampe1.is_on
```

```
True
```



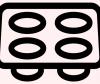
# RÉSUMÉ CLASSES

- Pour fabriquer des objets, on va créer un moule.  
En python, ce moule est ce qu'on appelle une **classe**.



# RÉSUMÉ CLASSES

- Pour fabriquer des objets, on va créer un moule.  
En python, ce moule est ce qu'on appelle une **classe**.
- Ce moule va permettre par la suite d'**instancier** des objets **différents** et **indépendants**.



# RÉSUMÉ CLASSES

- Pour fabriquer des objets, on va créer un moule.  
En python, ce moule est ce qu'on appelle une **classe**.
- Ce moule va permettre par la suite d'**instancier** des objets **différents** et **indépendants**.
- Une classe porte un **nom**, en majuscules;  
possède des **attributs**, déclarés dans une méthode `__init__`  
possède des **méthodes**, qui sont des fonctions de la classe  
produit des **instances**, qui sont des objets spécifiques créés avec le moule de la classe

# **ECRIRE DES CLASSES**

**</>**

# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

?

# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

**self** est un argument obligatoire dans toutes les méthodes des classes.

Il représente l'**instance** de la classe, donc l'**objet spécifique**.

C'est grâce à **self** qu'on va pouvoir accéder aux attributs et aux méthodes de la classe !

# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self, is_on):  
        # méthode spéciale : le constructeur  
        self.is_on = is_on  
  
    # définition des méthodes  
    def allumer(self):  
        self.is_on = True  
    def eteindre(self):  
        self.is_on = False
```

Pour accéder aux attributs et aux méthodes de nos classes,  
il suffit d'utiliser **self.<attribut ou méthode>**  
Ici, nous avons créé et manipulé l'attribut **is\_on** de la classe **Lampe**, qui permet de savoir si la lampe est allumée ou éteinte !

# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```

# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```



# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```



# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```

Pourquoi ne passe-t-on pas le paramètre self dans les méthodes de classe ?



# SELF, LA CLÉ DE VOUTE DE LA POO </>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

Le premier paramètre de **toutes** les méthodes de classe est **toujours** une référence vers l'instance courante de la classe !

Lors de l'appel d'une méthode, **Python s'occupe pour vous** de passer l'instance courante en paramètre !

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```

# SELF, LA CLÉ DE VOUTE DE LA POO <>

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

Le premier paramètre de **toutes** les méthodes de classe est **toujours** une référence vers l'instance courante de la classe !

Lors de l'appel d'une méthode, **Python s'occupe pour vous** de passer l'instance courante en paramètre !

```
l1 = Lampe()  
l1.allumer()  
l1.eteindre()
```



```
Lampe.allumer(l1)  
Lampe.eteindre(l1)
```

# ÉCRIRE LE CONSTRUCTEUR

```
class Lampe:  
    # définition des attributs  
    def __init__(self):  
        # méthode spéciale : le constructeur  
  
    # définition des méthodes  
    def allumer(self):  
        pass  
    def eteindre(self):  
        pass
```

Comme nous l'avons vu précédemment, **instancier** un objet appelle le **constructeur** de la classe, soit sa méthode **\_\_init\_\_**

Le rôle du constructeur est de définir les attributs de notre classe.

```
l1 = Lampe()  
l2 = Lampe()  
l3 = Lampe()
```

# ÉCRIRE LE CONSTRUCTEUR

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

# ÉCRIRE LE CONSTRUCTEUR

## Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on
```

# ÉCRIRE LE CONSTRUCTEUR

Lampe

~~taille : float~~  
~~poids : float~~  
~~couleur : str~~  
~~puissance: float~~  
~~is\_on: bool~~

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on
```

# ÉCRIRE LE CONSTRUCTEUR

## Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on
```

# ÉCRIRE LE CONSTRUCTEUR

## Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on
```

# ÉCRIRE LE CONSTRUCTEUR

## Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
class Lampe:  
    # définition des attributs  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on
```

```
lampe1 = Lampe(taille=60, poids=1350, couleur="noir", puissance=12, is_on=False)  
lampe2 = Lampe(taille=30, poids=850, couleur="rose", puissance=13.5, is_on=False)  
lampe3 = Lampe(taille=45, poids=1100, couleur="blanc", puissance=12, is_on=False)
```

# ÉCRIRE LES MÉTHODES

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()

eteindre()

```
class Lampe:  
    def __init__(self, taille, poids, couleur, puissance, is_on):  
        # méthode spéciale : le constructeur  
        self.taille = taille  
        self.poids = poids  
        self.couleur = couleur  
        self.puissance = puissance  
        self.is_on = is_on  
  
    # définition des méthodes  
    def allumer(self):  
        self.is_on = True  
    def eteindre(self):  
        self.is_on = False
```

# ÉCRIRE LES MÉTHODES

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
# définition des méthodes
def allumer(self):
    self.is_on = True

def eteindre(self):
    self.is_on = False
```

# ÉCRIRE LES MÉTHODES

Lampe

taille : float  
poids : float  
couleur : str  
puissance: float  
is\_on: bool

allumer()  
eteindre()

```
# définition des méthodes
def allumer(self):
    self.is_on = True

def eteindre(self):
    self.is_on = False
```

```
lampe1.is_on
False
lampe1.allumer()
lampe1.is_on
True
```



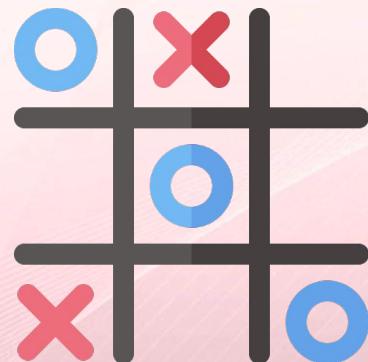
# EXERCICE

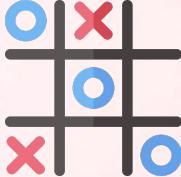
Créer un nouveau répertoire vide nommé *premiere\_classe* et l'ouvrir sur visual code.

Créer un fichier `main.py` et un fichier `lampe.py`

- Dans `lampe.py` : Créer la classe Lampe
- Dans `main.py` : Créer des instances de lampe en important la classe Lampe du fichier `lampe.py`

# PROJET MORPION





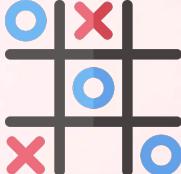
# PROJET MORPION

Créer un nouveau répertoire vide nommé **morpion** et l'ouvrir sur visual code.

Créer un fichier **main.py** et un fichier **morpion.py**

L'idée du projet est de créer une classe Morpion qui permet de jouer au jeu du morpion et qui est capable de déterminer le gagnant d'une partie.

Ensute, on entraînera une IA pour gagner au jeu du morpion



# PROJET MORPION

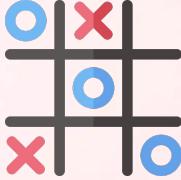
Créer une classe Morpion avec trois attributs :

- grille\_morpion
- tour\_jeu
- number\_moves\_played

Adapter les fonctions des exercices précédent en méthodes de la classe Morpion

## Méthodes à ramener :

affiche()  
get\_state()  
coup\_valide()  
position\_win()  
play\_move()  
play\_random\_move()



# PROJET MORPION

The screenshot shows a dark-themed code editor interface. On the left, the Explorer sidebar lists files: \_\_pycache\_\_, main.py, and morpion.py. The main editor area contains the following Python code:

```
from morpion import Morpion
morpion = Morpion()
morpion.affiche()
```

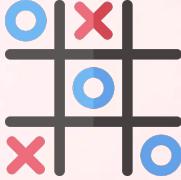
Below the editor, a navigation bar includes links for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL section displays command-line history:

- PS C:\Users\leosu\OneDrive\Documents\scripting\python> python .\morpion.py
- PS C:\Users\leosu\OneDrive\Documents\scripting\python> python .\main.py

Following this, a 3x3 grid of characters is shown:

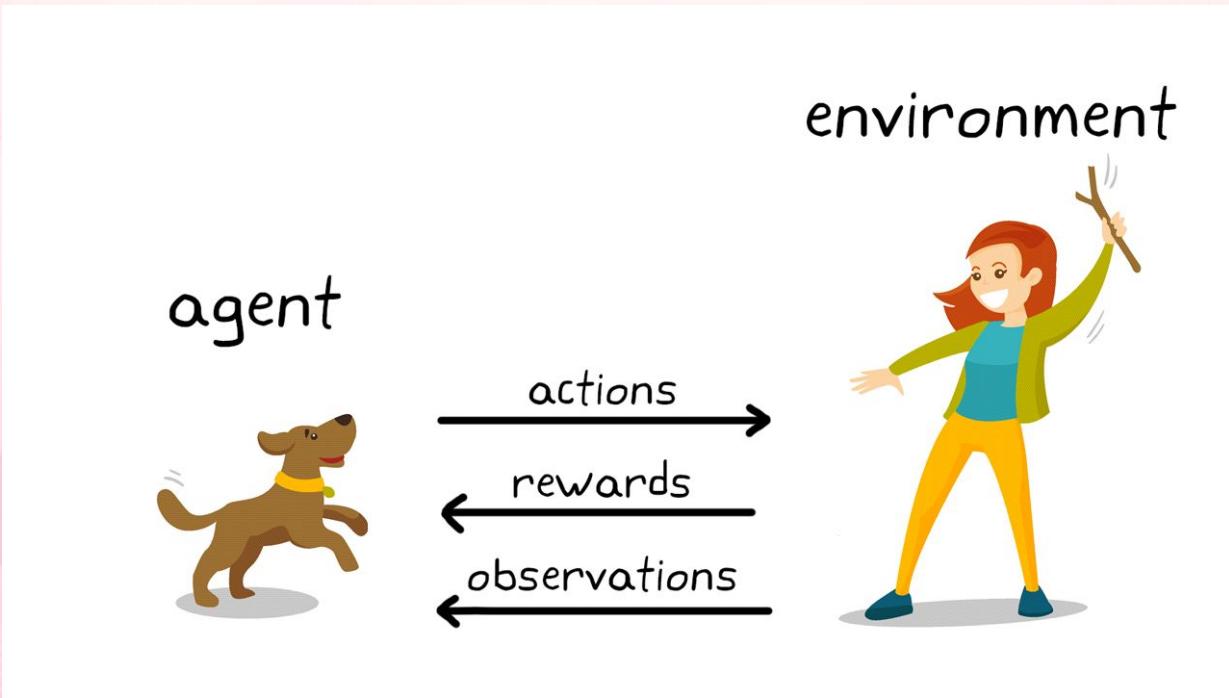
.	X	.
.	O	X
.	.	.

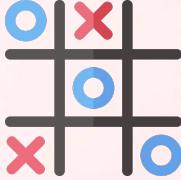
At the bottom of the terminal window, the prompt PS C:\Users\leosu\OneDrive\Documents\scripting\python> is visible.



# PROJET MORPION

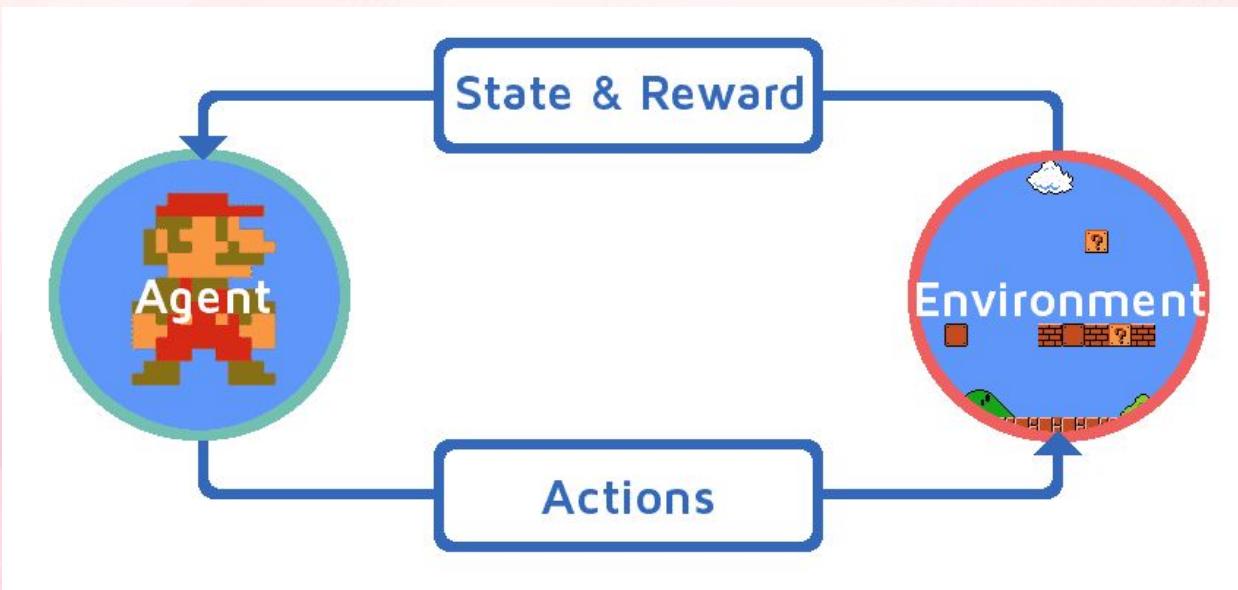
## Apprentissage renforcé

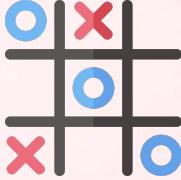




# PROJET MORPION

## Apprentissage renforcé

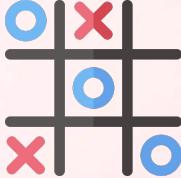




# PROJET MORPION

## Apprentissage renforcé

Pas besoin d'apprendre les règles du jeu à notre IA, on va juste l'entraîner avec le système de récompense ! Lorsqu'on fait un coup interdit ou perdant, on va la pénaliser. Lorsqu'elle gagne, on va la féliciter !

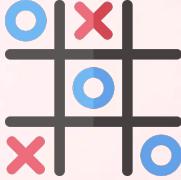


# PROJET MORPION

## Q-Learning

Faire une estimation du nombre d'états que peuvent prendre le jeu du morpion

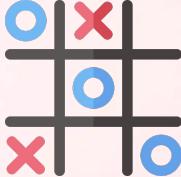
Quelles sont les actions possibles de notre agent?



# PROJET MORPION

## Q-Learning

actions = [(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2,1),(2,2)]

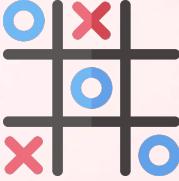


# PROJET MORPION

## Q-table

La Q table est un mapping entre des couples (state, action) vers des récompenses r.

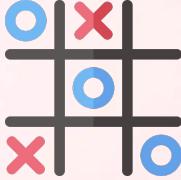
Cette table va permettre à notre agent de s'orienter dans ses décisions. Lorsqu'un état est gagnant, on va associer une récompense au couple (state, action). A l'inverse, lorsque l'état est perdant, on va associer une récompense négative à l'état



# PROJET MORPION

## Q-table

Q	
('....0....X.X', (2, 0))	: -10,
('....0....X.X', (2, 1))	: -9.17086638386959,
('....0....X.X', (2, 2))	: -3.2464612935222146,
('..0.0....XXX', (0, 0))	: -4.841500000000001,
('..0.0....XXX', (0, 1))	: -5.935000000000005,
('..0.0....XXX', (0, 2))	: -8.5,
('..0.0....XXX', (1, 0))	: -10,
('..0.0....XXX', (1, 1))	: -4.866500000000001,
('..0.0....XXX', (1, 2))	: -10,
('..0.0....XXX', (2, 0))	: 10,
('..0.0....XXX', (2, 1))	: -1.457008150000012,
('..0.0....XXX', (2, 2))	: -8.5,
('.....X..0', (0, 0))	: -7.746363107520936,
('.....X..0', (0, 1))	: -9.120030595968428,
('.....X..0', (0, 2))	: -7.689600401204338,
('.....X..0', (1, 0))	: -9.163286519004275,
('.....X..0', (1, 1))	: -9.12873073464024,
('.....X..0', (1, 2))	: -8.564630392274907,
('.....X..0', (2, 0))	: -10,



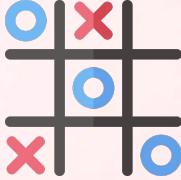
# PROJET MORPION

## Formule d'update du Q learning

$$Q(state, action) \leftarrow (1 - \alpha)Q(state, action) + \alpha \left( reward + \gamma \max_a Q(next\ state, all\ actions) \right)$$

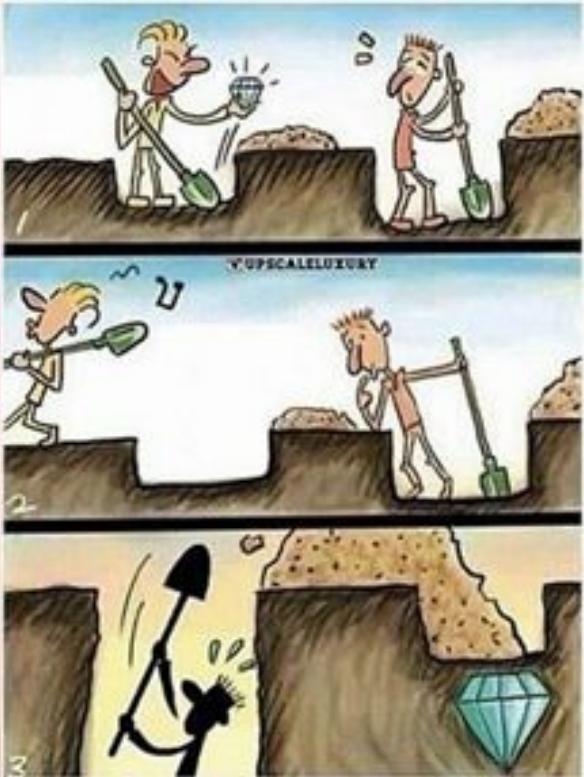
Where:

- $\alpha$  (alpha) is the learning rate ( $0 < \alpha \leq 1$ ) - Just like in supervised learning settings,  $\alpha$  is the extent to which our Q-values are being updated in every iteration.
- $\gamma$  (gamma) is the discount factor ( $0 \leq \gamma \leq 1$ ) - determines how much importance we want to give to future rewards. A high value for the discount factor (close to 1) captures the long-term effective award, whereas, a discount factor of 0 makes our agent consider only immediate reward, hence making it greedy.

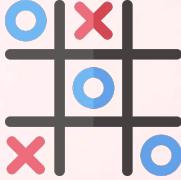


# PROJET MORPION

## Exploration exploitation ?

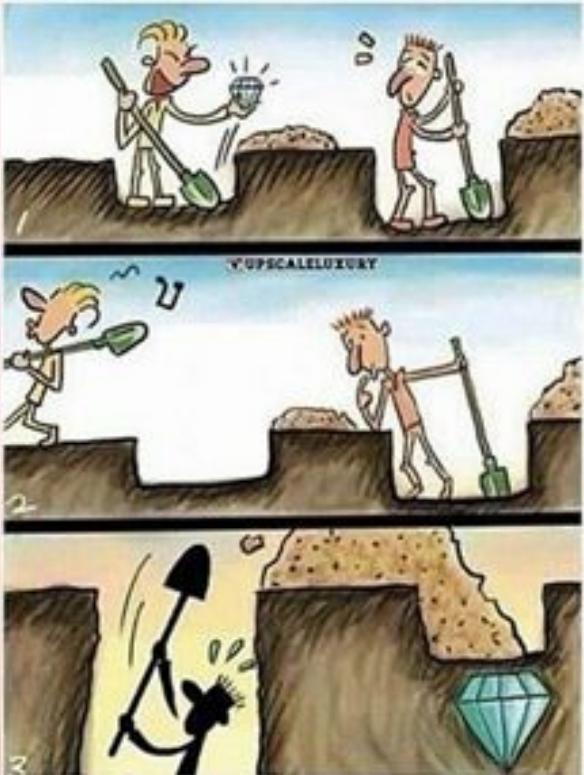


```
while True:  
    if np.random.uniform(0,1) < exploration_proba:  
        action = m.random_move()  
    else:  
        state = m.get_state()  
        action = get_best_next_move(state, Q)
```



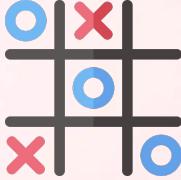
# PROJET MORPION

## Exploration exploitation ?



```
while True:  
    EXPLO if np.random.uniform(0,1) < exploration_proba:  
          action = m.random_move()  
    else:  
        EXPLOIT state = m.get_state()  
              action = get_best_next_move(state, Q)
```

Q
(...o...X.X', (2, 0)): -10, (...o...X.X', (2, 1)): -9.17086638386959, (...o...X.X', (2, 2)): -3.2464612935222146, ('.o.o...XXX', (9, 0)): -4.841500000000001, ('.o.o...XXX', (9, 1)): -5.935000000000005, ('.o.o...XXX', (9, 2)): -8.5, ('.o.o...XXX', (1, 0)): -10, ('.o.o...XXX', (1, 1)): -4.866500000000001, ('.o.o...XXX', (1, 2)): -10, ('.o.o...XXX', (2, 0)): 10, ('.o.o...XXX', (2, 1)): -1.4570081500000012, ('.o.o...XXX', (2, 2)): -8.5, ('....X..O', (9, 0)): -7.74633107520936, ('....X..O', (9, 1)): -9.12003059596428, ('....X..O', (9, 2)): -7.689600401204338, ('....X..O', (1, 0)): -9.163286519004275, ('....X..O', (1, 1)): -9.12873073464024, ('....X..O', (1, 2)): -8.564630392274907, ('....X..O', (2, 0)): -10,

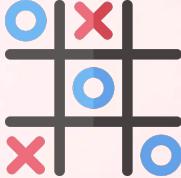


# PROJET MORPION

## Exploration exploitation ?



RANDOM VS QTABLE



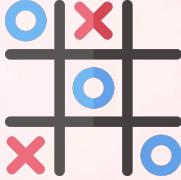
# PROJET MORPION

Techniques d'update de la Q table

SI l'agent gagne la partie : reward = 10

SI l'agent fait égalité : reward = 0

SI l'agent perd la partie : reward = -10



# PROJET MORPION

Objectif :

Entraîner sur 1 million d'itérations notre agent Q learning contre une ia aléatoire et voir le résultat.