

# PENSÉE COMPUTATIONNELLE ET PROGRAMMATION

## 2<sup>ème</sup> INFO2

Enseignante: Mme Houda Ben Saïd Mangour

Séance : 03/10/2022

# OBJECTIFS DE LA SÉANCE

## 1. Tester les solutions des sous-problèmes



➤ Implémenter l'algorithme en python

Calculatrice

P1:  
 $A+B$ ,  $A-b$  et  $n^2$

P1\_1:  
 $A+B$  et  $A-b$

P1-1a

P1\_1a  
 $A+B$

P1\_1b  
 $A-B$

P1-1b

## ALGORITHME P1-1a\_addition

P1-1a

DÉBUT

```
Ecrire("Donner la valeur de A1:")
Lire(A1)
Ecrire("Donner la valeur de B1:")
Lire(B1)
    Add ← A1+B1
Ecrire("La somme de",A1, "et de",
B1,"est:",Add)
```

FIN

T.D.O:

Objets	Types
A1	Entier
B1	Entier
Add	Entier

## ALGORITHME P1-1b\_soustraction

P1-1b

DÉBUT

```
Ecrire("Donner la valeur de A2:")
Lire(A2)
Ecrire("Donner la valeur de B2:")
Lire(B2)
    S ← A2-B2
Ecrire("La soustraction de",A2, "et de",
B2,"est:", S)
```

FIN

T.D.O:

Objets	Types
A2	Entier
B2	Entier
S	Entier


A stick figure is sitting at a desk, holding a pen to its chin in a thinking pose. A thought bubble above its head contains the text "Algorithme réalisé veut dire solution trouvée".

Algorithme réalisé  
veut dire solution  
trouvée

Algorithme  
=  
Solution

A stick figure is sitting at a desk, looking stressed with sweat drops on its head. It is holding a pen to its chin. A thought bubble above its head contains the text "Comment vérifier si la solution trouvée est correcte?".

Comment vérifier si  
la solution trouvée  
est correcte?

A stick figure is sitting at a desk, holding a pen to its chin. A thought bubble above its head contains the text "On doit implémenter l'Algorithme en un langage de programmation".

On doit implémenter  
l'Algorithme en un  
langage de  
programmation

CODIFICATION

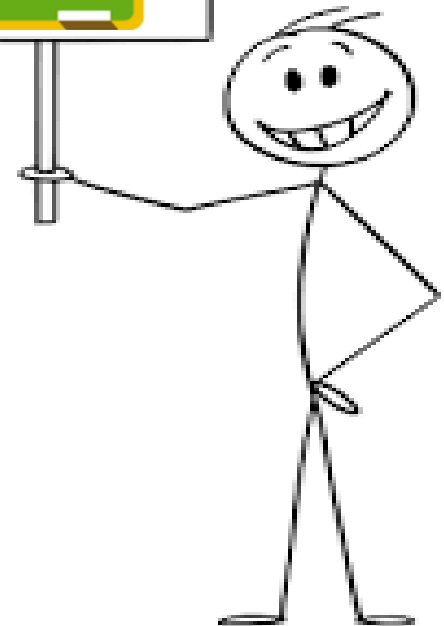


# Activité (CODIFICATION)

Essayer de réécrire l'algorithme avec un langage compréhensible par l'ordinateur (Python)



Vous pouvez vous servir du  
**Mémento\_modifié** publié dans  
Classroom



# Mémento\_1

## Blocs d'instructions

instruction parente :

bloc d'instructions 1...

⋮

instruction parente :

bloc d'instructions 2...

⋮

instruction suivante après bloc 1

🔧 régler l'éditeur pour insérer 4 espaces à la place d'une tabulation d'indentation.

entier, flottant, booléen, chaîne, octets

## Types de base

```
int 783 0 -192 0b010 0o642 0xF3
      zéro      binaire      octal      hexa
float 9.23 0.0 -1.7e-6
bool True False
str "Un\nDeux"
      retour à la ligne échappé
      'L\''âme'
      ' échappé
bytes b"toto\xfe\775"
      hexadecimal      octal
```

Chaîne multiligne :

```
"""X\tY\tZ
1\t2\t3"""
```

tabulation échappée

🔧 immutables

## Identificateurs

pour noms de variables,  
fonctions, modules, classes...

**a...zA...Z\_** suivi de **a...zA...Z\_0...9**

- ☐ accents possibles mais à éviter
- ☐ mots clés du langage interdits
- ☐ distinction casse min/MAJ

☺ a toto x7 y\_max BigOne

☹ 8y and for

## = Variables & affectation

🔧 affectation ⇔ **association** d'un nom à une valeur

- 1) évaluation de la valeur de l'expression de droite
- 2) affectation dans l'ordre avec les noms de gauche

**x=1.2+8+sin(y)**

**a=b=c=0** affectation à la même valeur

**y,z,r=9.2,-7.6,0** affectations multiples

**a,b=b,a** échange de valeurs

**a,\*b=seq** } dépaquetage de séquence en  
**\*a,b=seq** } élément et liste

**x+=3** incrémentation ⇔ **x=x+3**

**x-=2** décrémentation ⇔ **x=x-2**

**x=None** valeur constante « non défini »

**del x** suppression du nom **x**

et

**\***

**/**

**%**

...

```
print("v=", 3, "cm :", x, ", ", y+4)
```

éléments à afficher : valeurs littérales, variables, expressions

Options de **print**:

- ☐ **sep=" "** séparateur d'éléments, défaut espace
- ☐ **end="\n"** fin d'affichage, défaut fin de ligne
- ☐ **file=sys.stdout** print vers fichier, défaut sortie standard

**s = input("Directives:")**

🔧 **input** retourne toujours une **chaîne**, la convertir vers le type désiré (cf. encadré *Conversions* au recto).

## Affichage

## Saisie

# Correction



**Si** vous avez réussi à utiliser le memento, vous devez avoir un script Python qui ressemble à celui-ci:

```
1 A1=input("Donner la valeur de A:")
2 B1=input("donner la valeur de B:")
3 Add=A1+B1
4 print("la somme de A et de B est:",Add)
```

## **Sinon**

- ✓ Lancez l'éditeur Python de votre choix
- ✓ Recopier le script
- ✓ Enregistrez-le sous le nom « somme » dans votre dossier personnel.
- ✓ Exécuter ce programme: pour A: 2  
pour B: 3



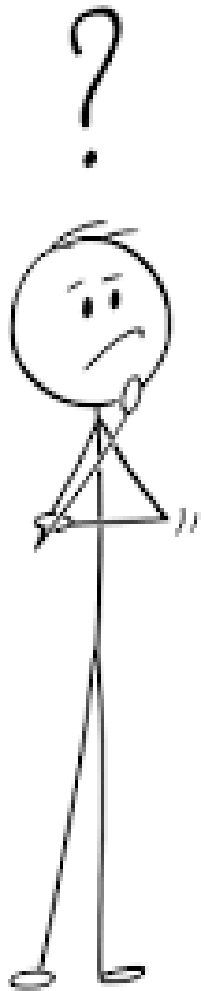
# Aperçu de l'exécution



En cours d'exécution: SOMME.py

```
Donner la valeur de A:2  
donner la valeur de B:3  
la somme de A et de B est: 23  
>>> |
```





- ✓ Pourquoi le résultat est erroné?
- ✓ Que s'est il passé?
- ✓ Pourquoi il y'a eu concaténation des deux valeurs ?
- ✓ Quel est le rôle de l'instruction « `input( )` »?
- ✓ Quel est le rôle de l'instruction « `print( )` »?

✓ Pour comprendre le fonctionnement de la fonction `input( )`, on va utiliser la fonction « **Type(Nom\_variable)** » qui permet de déterminer le **type du contenu** des variables A1 et B1

```
addition.py x
1 A1=input("Donner la valeur de A:")
2 B1=input("donner la valeur de B:")
3 #On veut voir le type des valeurs saisies
4 #On utilise la fonction Type(Nom_variable) comme suit:
5 print("le type de la valeur de la variable A1 est:",type(A1))
6 print("le type de la valeur de la variable B1 est:",type(B1))
```



```
>>> %Run addition.py
```

```
Donner la valeur de A:2
```

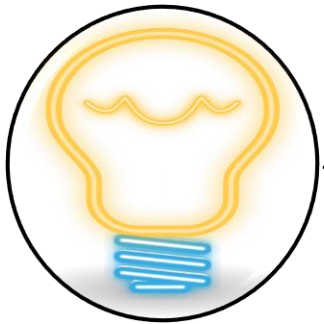
```
donner la valeur de B:3
```

```
le type de la valeur de la variable A1 est: <class 'str'>
```

```
le type de la valeur de la variable B1 est: <class 'str'>
```

```
>>>
```

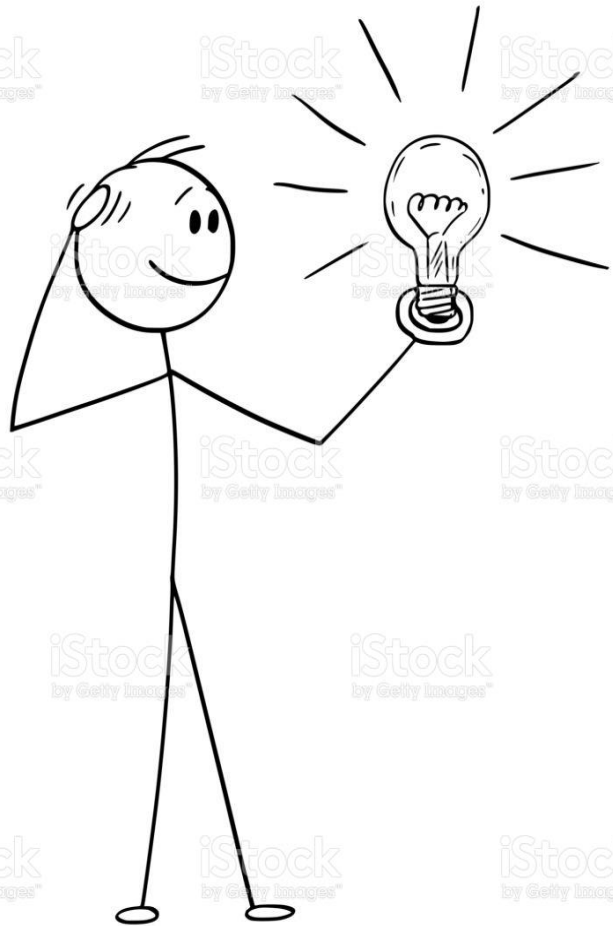
# Constatation:



Les valeurs affectées aux variables par l'instruction `input()` sont de type `string` (chaîne de caractères)

✓ Ça explique le résultat obtenu lors de l'exécution du programme de l'addition: l'opérateur `+` a joué le rôle d'opérateur de concaténation au lieu d'opérateur d'addition.

# Que faire?



➔ Convertir la valeur saisie au clavier suite à l'utilisation de la fonction **input()** selon le type voulu

**Si** on veut que la variable saisie soit de type Entiers **Alors**

Nom\_variable = **int**(input('message à afficher'))

**Sinon**

**Si** on veut que la variable saisie soit de type réel **Alors**

Nom\_variable = **float**(input('message à afficher'))

**SINON**

Nom\_variable = input('message à afficher')





✓ Dans notre cas, l'ajout la fonction `int()` a permis de convertir les chaînes de caractères en des entiers.

```
1 A1=input("Donner la valeur de A:")
2 B1=input("donner la valeur de B:")
3 Add=A1+B1
4 print("la somme de A et de B est:",Add)
5
```

**Avant** l'ajout de  
`int()`

En cours d'exécution: SOMME.py

```
Donner la valeur de A:2
donner la valeur de B:3
la somme de A et de B est: 23
>>> |
```

```
1 A1=int(input("Donner la valeur de A:"))
2 B1=int(input("donner la valeur de B:"))
3 Add=A1+B1
4 print("la somme de A et de B est:",Add)
5
```

**Après** l'ajout de  
`int()`

En cours d'exécution: SOMME.py





```
Donner la valeur de A:3
donner la valeur de B:2
la somme de A et de B est: 5
>>> |
```

à noter!

Les instructions `input( )` et `print( )` permettent de "dialoguer" avec une machine, elles s'appellent les instructions "**d'entrée/sortie**" ou de "**lecture/écriture** "

# Recap



	<b>Déclaration</b> 	
<b>Algorithme</b>	Lire (A1)	Lire (B1)
<b>Python</b>	<code>A1=int(input("Donner la valeur de A:"))</code>	<code>B1=int(input("donner la valeur de B:"))</code>
		
<b>Algorithme</b>	<b>Affectation</b>	$Add \leftarrow A1 + B1$
<b>Python</b>		<code>Add=A1+B1</code>
		
<b>Algorithme</b>	Ecrire("la somme de A et de B est:",Add)	
<b>Python</b>	<code>print("la somme de A et de B est:",Add)</code>	



LESSONS

LEARNED





“  
**Les structures simples**  
”

**Les opérations d'entrée**

**Les opérations de sortie**

**Les opérations d'affectation**