Introduction à l’informatique CM2

Antonio E. Porreca
aeporreca.org/introinfo

Algorithmes !

C’est quoi un algorithme ?

- La description non ambiguë d’une séquence finie d’instructions permettant de résoudre un problème

- Finitude = termine après un nombre fini d’étapes

- Non ambigu = précis (en termes d’opérations élémentaires)

- Entrées = données

- Sorties = résultat attendu

محمد بن موسى الخوارزمي

Muḥammad ibn Mūsā al-Khwārizmī,
auteur de الجبر (al-Jabr)

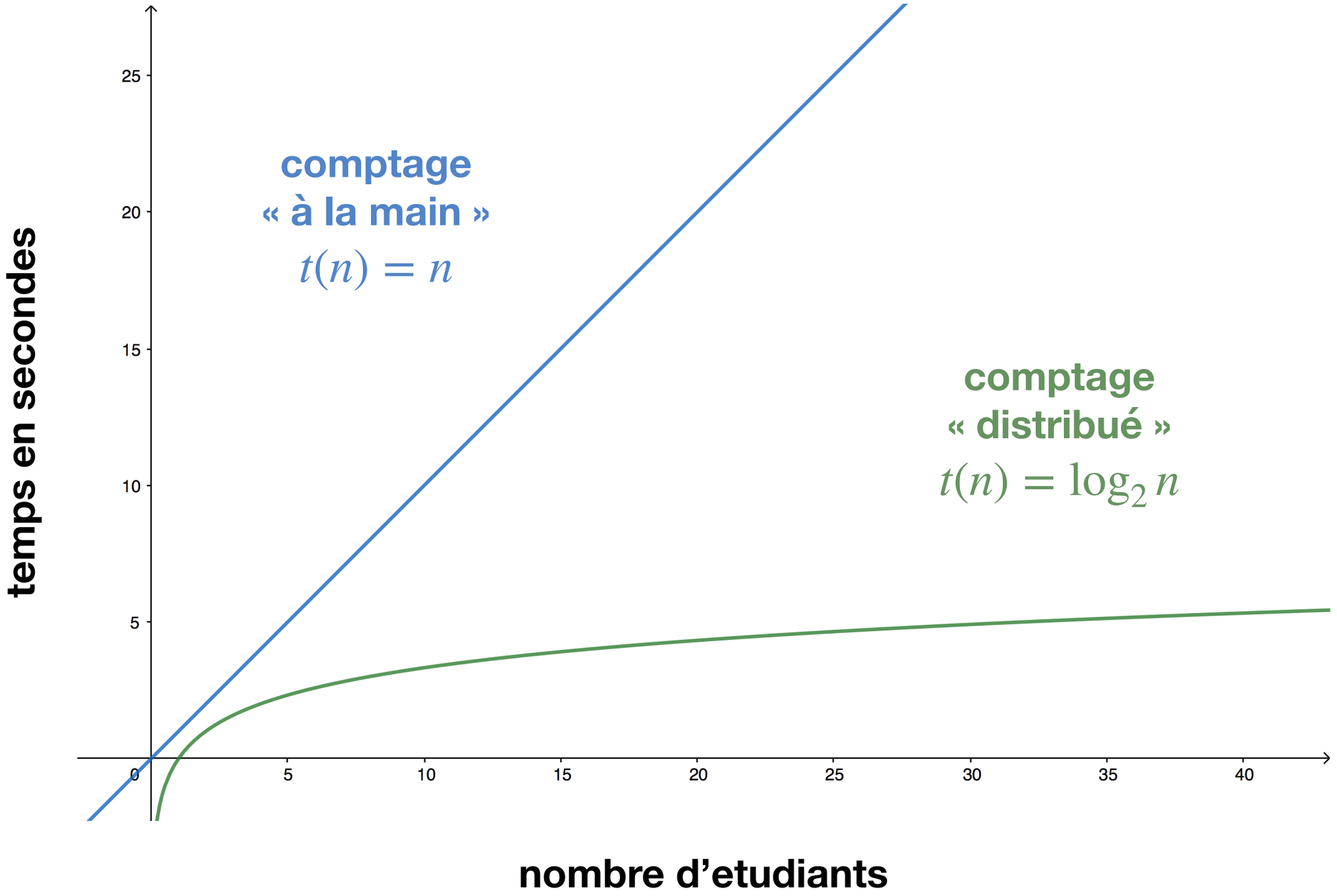
Combien d’étudiants y a-t-il dans la salle ?

Peut-on faire mieux que l’algo naif si on collabore ?

Combien d'étudiants y a-t-il dans la salle ?

- Chaque étudiant commence avec le nombre 1 en tête
- **Tant qu'**il reste au moins deux étudiants debout :
    - Chaque étudiant encore debout cherche du regard un autre étudiant debout
    - Les deux étudiants s'échangent le nombre qu'ils ont en tête
    - L'un des deux étudiants s'assoit
    - L'autre additionne les deux nombres qu'il mémorise
- Le dernier étudiant debout crie le nombre qu'il a en tête

Efficacité du comptage



Résoudre un problème

- On cherche un algorithme
- On le décrit précisément, de manière non ambigüe
- On prouve qu'il est correct
- On vérifie qu'il est efficace (idéalement, on choisit l'algorithme optimal)
- On le met en œuvre (pas dans cette UE)
- On le teste (pas dans cette UE)

Décrire des algorithmes

- En langage naturel (par exemple, en français avec un accent italien)

- En pseudocode (semi-formel)

- En langage de programmation (formel)

    - Par exemple, en Python comme ici et dans l'UE Mise en œuvre informatique

Recherche dans une séquence en langage naturel

- Pour chaque élément de la séquence à partir du premier :
    - Si cet élément est l'élément cherché, on a terminé
    - Sinon, on continue avec l'élément suivant
- S'il n'y a plus d'éléments et on n'a pas trouvé ce qu'on cherchait, alors il n'est pas là

Recherche dans une séquence en pseudocode

pour chercher *élément* **dans** *séquence*
    soit n = longueur(séquence)
    soit $i = 0$
    tant que $i < n$ **faire**
        si l'i-ème element de *séquence* est *élément* **alors**
            résultat i
        incrémenter i
    résultat -1

Recherche dans une séquence en Python 🐍

```
def chercher(element, sequence):
    n = len(sequence)
    i = 0
    while i < n:
        if sequence[i] == element:
            return i
        i = i + 1
    return -1
```

Structures de contrôle

```
instruction1
instruction2
…
instructionn
```

```
if condition:
    instructions
else:
    d'autres instructions
```

```
while condition:
    instructions
```

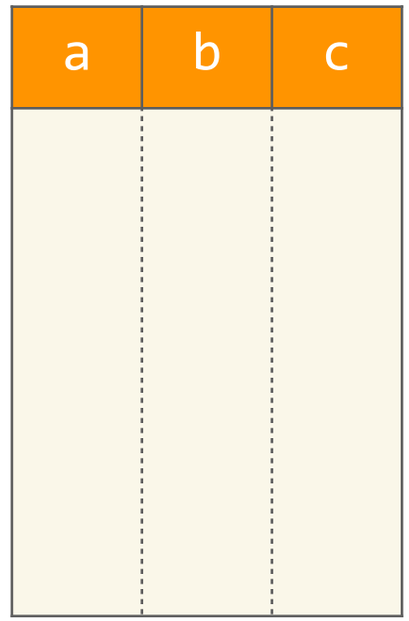
Algorithmes avec des séquences d’instructions

Exécution d'une séquence d'instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```

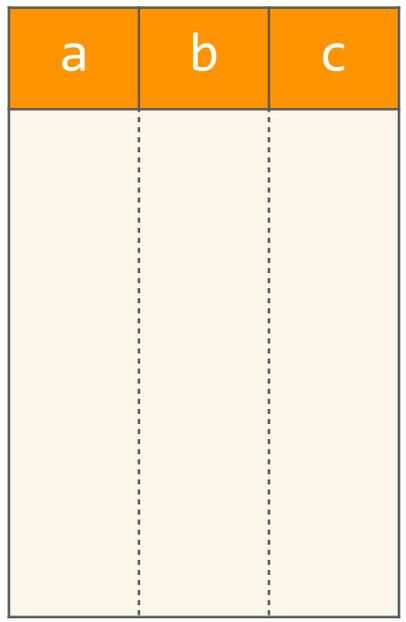
Exécution d’une séquence d’instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```



Exécution d’une séquence d’instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```



Exécution d’une séquence d’instructions

```
    a = 1
👉  b = 5
    c = a + b
    a = 3
    b = c - a
```

a	b	c
1		

Exécution d'une séquence d'instructions

```
    a = 1
    b = 5
👉  c = a + b
    a = 3
    b = c - a
```

a	b	c
1		
	5	

Exécution d'une séquence d'instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```

👉

a	b	c
1		
	5	
		6

Exécution d'une séquence d'instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```

👉

a	b	c
1		
	5	
		6
3		

Exécution d'une séquence d'instructions

```
a = 1
b = 5
c = a + b
a = 3
b = c - a
```

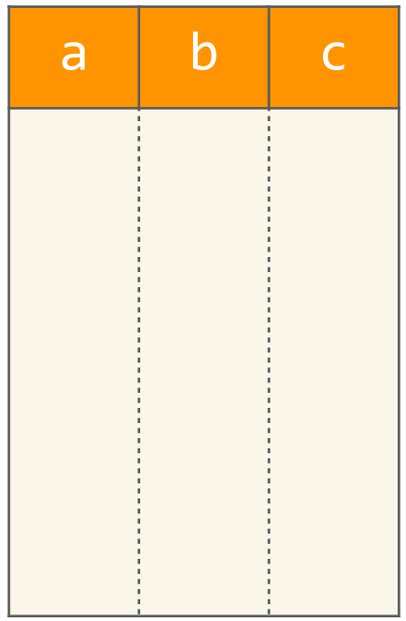
a	b	c
1		
	5	
		6
3		
	3	

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```



Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

a	b	c

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

a	b	c
1		

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

👉

a	b	c
1		
	5	

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

👉

a	b	c
1		
	5	
3		

Un algorithme avec une erreur

```
a = 1
b = 5
a = 3
b = c - a
```

a	b	c
1		
	5	
3		
	⚠	

erreur !

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(1, 3, 2)
```

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉 `>>> discriminant(1, 3, 2)`

Définition de fonctions : discriminant de $ax^2 + bx + c$

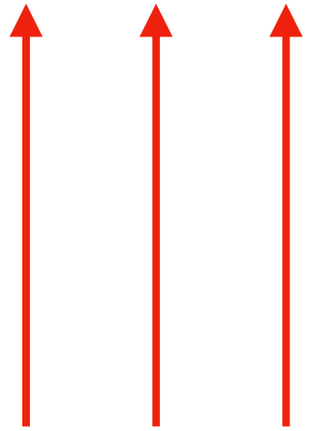
```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉 `>>> discriminant(1, 3, 2)`

a	b	c	d	e	delta

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```



a	b	c	d	e	delta

👉 `>>> discriminant(1, 3, 2)`

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉 `>>> discriminant(1, 3, 2)`

a	b	c	d	e	delta
1	3	2			

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(1, 3, 2)
```

a	b	c	d	e	delta
1	3	2			

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉

```
>>> discriminant(1, 3, 2)
```

a	b	c	d	e	delta
1	3	2	9		

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
👉  delta = d - e
    return delta

>>> discriminant(1, 3, 2)
```

a	b	c	d	e	delta
1	3	2			
			9		
				8	

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

```
>>> discriminant(1, 3, 2)
```

a	b	c	d	e	delta
1	3	2	9	8	1

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(1, 3, 2)
1
```

a	b	c	d	e	delta
1	3	2	9	8	1

Appel avec d'autres entrées :

$$5x^2 + x + 1$$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

👉 >>> discriminant(5, 1, 1)
```

Définition de fonctions : discriminant de $ax^2 + bx + c$

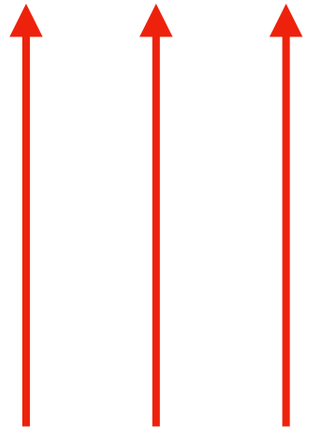
```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

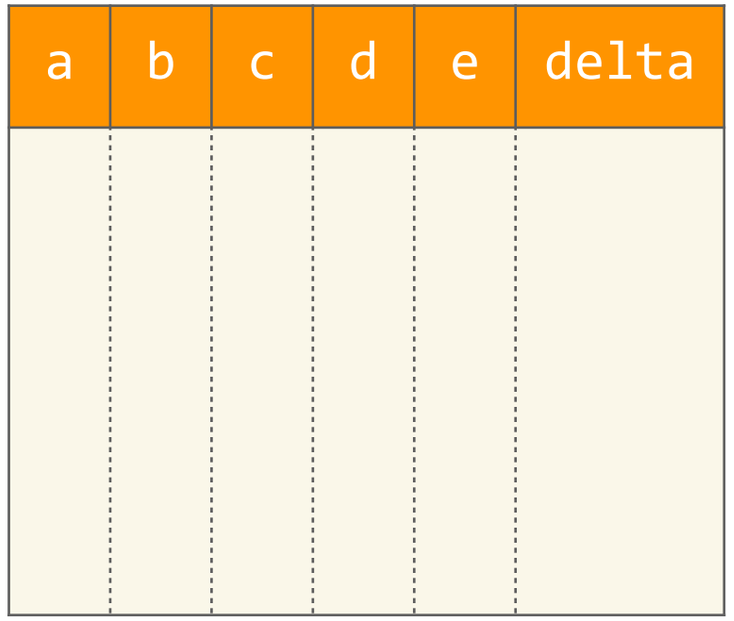
👉 `>>> discriminant(5, 1, 1)`

a	b	c	d	e	delta

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

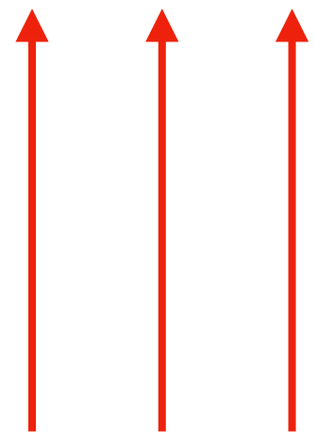





```
>>> discriminant(5, 1, 1)
```

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```



a	b	c	d	e	delta
5	1	1			

👉 `>>> discriminant(5, 1, 1)`

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(5, 1, 1)
```

a	b	c	d	e	delta
5	1	1			

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉

```
>>> discriminant(5, 1, 1)
```

a	b	c	d	e	delta
5	1	1	1		

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(5, 1, 1)
```

a	b	c	d	e	delta
5	1	1	1	20	

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta
```

👉

```
>>> discriminant(5, 1, 1)
```

a	b	c	d	e	delta
5	1	1	1	20	-19

Définition de fonctions : discriminant de $ax^2 + bx + c$

```
def discriminant(a, b, c):
    d = b * b
    e = 4 * a * c
    delta = d - e
    return delta

>>> discriminant(5, 1, 1)
-19
```

👉

a	b	c	d	e	delta
5	1	1	1	20	-19

Algorithmes avec des conditions

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
>>> solutions(1, 3, 2)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
👉 >>> solutions(1, 3, 2)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2

👉 `>>> solutions(1, 3, 2)`

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

a	b	c	delta	x1	x2
1	3	2			

```
>>> solutions(1, 3, 2)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

a	b	c	delta	x1	x2
1	3	2	1		

```
>>> solutions(1, 3, 2)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

```
>>> solutions(1, 3, 2)
```

a	b	c	delta	x1	x2
1	3	2	1		

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

```
>>> solutions(1, 3, 2)
```

a	b	c	delta	x1	x2
1	3	2	1	-1.0	

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	3	2	1	-1.0	-2.0

```
>>> solutions(1, 3, 2)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	3	2	1	-1.0	-2.0

```
>>> solutions(1, 3, 2)
[-1.0, -2.0]
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
👉 >>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2

👉 `>>> solutions(5, 1, 1)`

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

a	b	c	delta	x1	x2
5	1	1			

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
5	1	1	-19		

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
5	1	1	-19		

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
5	1	1	-19		

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
5	1	1	-19		

👉

```
>>> solutions(5, 1, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
5	1	1	-19		

```
>>> solutions(5, 1, 1)
[]
```

👉

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

```
👉 >>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2

👉 `>>> solutions(1, 2, 1)`

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

a	b	c	delta	x1	x2
1	2	1			

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

👉

a	b	c	delta	x1	x2
1	2	1	0		

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	2	1	0		

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	2	1	0		

👉

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
👉      return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	2	1	0	-1.0	

```
>>> solutions(1, 2, 1)
```

Solutions de $ax^2 + bx + c = 0$

```
def solutions(a, b, c):
    delta = discriminant(a, b, c)
    if delta > 0:
        x1 = (-b + sqrt(delta)) / (2 * a)
        x2 = (-b - sqrt(delta)) / (2 * a)
        return [x1, x2]
    elif delta == 0:
        x1 = -b / (2 * a)
        return [x1]
    else:
        return []
```

a	b	c	delta	x1	x2
1	2	1	0	-1.0	

```
>>> solutions(1, 2, 1)
[-1.0]
```

Algorithmes avec de l’itération

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

Somme des premiers n entiers

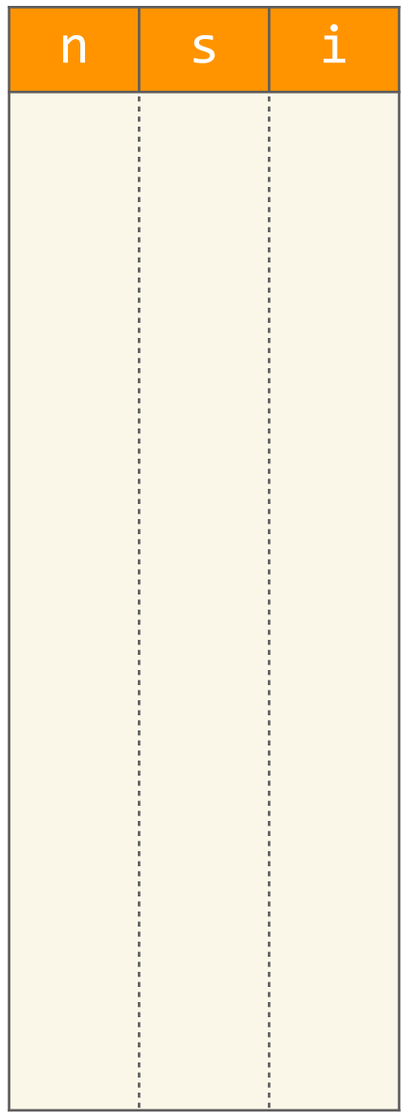
```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉 `>>> somme(10)`

Somme des premiers n entiers

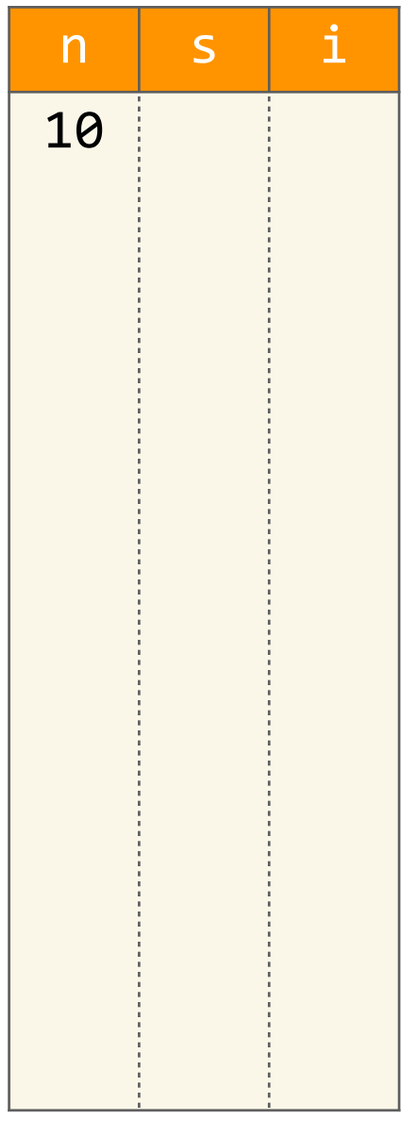
```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉 `>>> somme(10)`



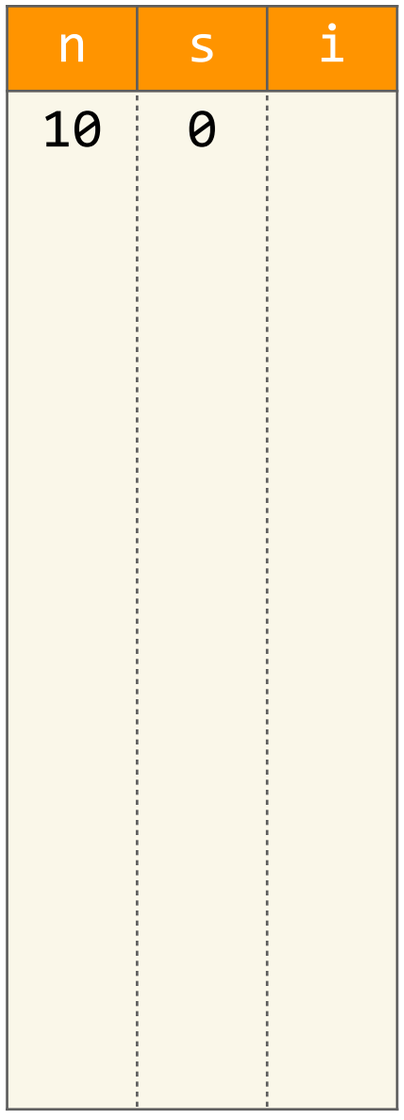
Somme des premiers n entiers

```
    def somme(n):
👉      s = 0
        i = 1
        while i <= n:
            s = s + i
            i = i + 1
        return s

>>> somme(10)
```



Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```



Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉

```
>>> somme(10)
```

n	s	i
10	0	1

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉

```
>>> somme(10)
```

n	s	i
10	0	1

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
		2

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

```
>>> somme(10)
```

n	s	i
10	0	
	1	1
		2

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2

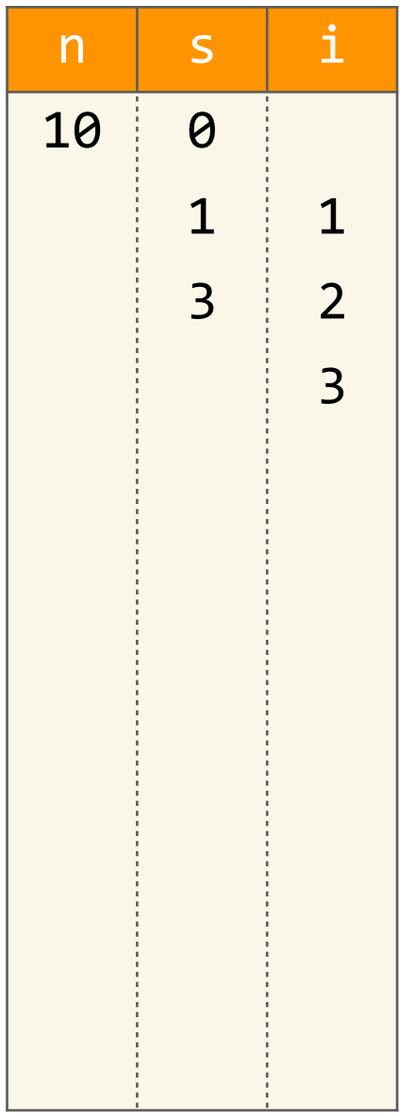
Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
		3

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```



n	s	i
10	0	
	1	1
	3	2
		3

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
		4

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
		4

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
		5

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
		5

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
		6

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
		6

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
		7

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
		7

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
		8

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
		8

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
		9

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
		9

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
		10

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉

```
>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
		10

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
	55	10

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
👉  while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
	55	10
		11

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s
```

👉

```
>>> somme(10)
```

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
	55	10
		11

Somme des premiers n entiers

```
def somme(n):
    s = 0
    i = 1
    while i <= n:
        s = s + i
        i = i + 1
    return s

>>> somme(10)
55
```

👉

n	s	i
10	0	
	1	1
	3	2
	6	3
	10	4
	15	5
	21	6
	28	7
	36	8
	45	9
	55	10
		11

Algorithmes avec de l'interaction avec un utilisateur ou une utilisatrice

Resolution interactive d’équations de 2eme degré

```
def résoudre_equation():
    print("Bonjour, je vais résoudre ax² + bx + c = 0")
    print("Tu t'appelles comment ?")
    nom = input()
    print("Donne-moi la valeur de a :")
    a = float(input())
    print("Donne-moi la valeur de b :")
    b = float(input())
    print("Donne-moi la valeur de c :")
    c = float(input())
    sol = solutions(a, b, c)
    print(nom, "voici les solutions :", sol)
```