Formulaire Python

Spébio 2

juin 2026

Table des matières

Remarques

Le symbole ⛰ indique les parties que vous pouvez omettre lors de la première lecture de ce document, .

Les flèches ⟶ qui apparaissent dans les programmes servent à visualiser les indentations à respecter pour structurer les blocs.

Utilisation dans un navigateur internet

En allant sur le site https://replit.com vous pouvez créer un compte et programmer en python. La plupart des modules sont disponibles.

Installation ⛰

Si vous voulez obtenir une installation d'anaconda une distribution de Python qui permet de travailler sur de plus grands programmes vous trouverez la procédure d'installation à l'adresse https://pyzo.org/start.html. Vous devez suivre la procédure indiquée et choisir comme interpréteur **miniconda** et comme version de python la **3.***.

I Opérations

I.1 Sur les réels

Les opérations usuelles sont :

1. `x * y`
   produit de x par y
2. `x / y`
   quotient de x par y
3. `x + y`
   somme de x et y
4. `x - y`
   différence de x et y
5. `x ** y`
   x puissance y
6. `abs(x)`
   valeur absolue.

Si on a besoin de fonctions mathématiques, même basiques comme sinus, cosinus, racine carrée, il faut les *importer* d'un *module* (une collection de fonction.

```
>>>import math

>>>math.sqrt(5)
>>>2.23606797749979
```

ou

```
>>>from math import sqrt
>>>sqrt(5)
2.23606797749979
```

Remarque : Si on a chargé le module `maths` les constantes usuelles sont alors utilisables. Par exemple `math.pi`, `math.e`.

I.2 Variables et types : une courte introduction

Une *Variable* est un nom associé à un type et à une valeur. Elles nous permettent d'enregistrer des valeurs pour les intégrer dans les calculs.

```
>>>x=1
>>>y=x+1
>>>print(y)
```

La valeur affichée est 2.

L'action de lier une valeur à un nom est *l'affectation*. Elle l'utilise l'opérateur =. L'expression à droite de l'opérateur est *évalueée* et liée au nom placé à gauche du signe égale. Les égalités se lisent donc de droite à gauche, en pseudo langage elles sont notées par ←.

Python nous permet de faire des affectations parallèles

```
>>>n,x,y= 52,1.5,"bonjour"
```

La plus part des variables peuvent voir leur valeur modifiée.

```
>>>x=1
>>>x=x+1
>>>print(x)
2
```

Les variables ont un *type* : entier, chaîne de caractères, nombre à virgule flottante. On peut connaître le type d'une variable avec la commande `type`.

```
n,x,l= 52,1.5,"bonjour"
type(n),type(x),type(l)
(<class 'int'>, <class 'float'>, <class 'str'>)
```

Nous préciserons dans les cours et TP suivants cette notion, mais nous pouvons déjà retenir que `int` désigne un entier (anglais *integer*),`float` désigne un nombre à virgule et `str` une chaine de caractères (anglais *string*), `list` une liste.

I.3 Opérations arithmétiques

Reprenons les opérations précédentes.

```
>>> 2/1
2.0

>>> 2/3
0.6666666666666666
```

```
>>> 2+0
2

>>> 2+1.0
3.0
```

On devine que 2.0 désigne un nombre réel et 2 un nombre entier. Nous verrons dans les prochains TP comment Python gère ces différents *types* de données.

Lorsque on effectue une opération entre deux entiers et que l'opération est compatible avec les entiers on obtient un résultat de type entier, sinon on obtient un résultat réel.

Exercice 1.
Quelles sont les opérations compatibles avec le type entier?

Lorsque l'on veut travailler avec des entiers, par exemple pour faire de l'arithmétique, on peut utiliser deux autres opérations

1. `x // y`

   quotient dans la division euclidienne de x par y.

2. `x % y`

   reste dans la division euclidienne de x par y.

```
>>> 3//2
1

>>> 3%2
1
```

À retenir

Les opérations sur les entiers et sur les réels ne sont pas les mêmes.
Il faut distinguer la division réelle opérateur / du calcul du quotient dans la division euclidienne opérateur //.

I.4 Opérations sur les chaînes de caractères

Une *chaîne de caractères* est un mot, une phrase, une suite de caractères. Elle est délimité par des apostrophe (' ') ou des guillemets (" ").

1. `'chaine1' + 'chaine2'`

   concaténation rassemble deux chaines met la deuxième à la suite de la première.

2. `'chaine' *n`

   repète *n* fois la chaine.

```
>>> 'bonjour'+' au revoir'
'bonjour au revoir'

>>> "beaucoup "*3
'beaucoup beaucoup beaucoup '
```

II Fonctions

II.1 Généralités

```
from math import*

def hypothenus(a,b):
        h=sqrt(a**2+b**2)
        return  h
```

II.2 Fonction sans `return`, sans argument, arguments par défaut .

Il est tout à fait possible d'avoir une fonction sans `return`, qui quand même un *effet*.

```
## exemple fonction sans return
def repete(chaine,n):
        r=chaine * n
        print(r)
```

```
## exemple fonction , argument par défaut
def repete(chaine,n=2):
        r=chaine * n
        print(r)
```

```
## fonction ni argument ni retour
def salutation():
        print('bonjour tout le monde')
```

II.3 Portée des variables

On peut définir des variables à plusieurs endroits, dans le corps principal du programme, dans les fonctions...

La *portée d'un identifiant* est l'étendue où l'on peut utiliser la valeur liée à identifiant.

```
a= 1
def plusa(x):
        return a+x

print(plusa(10))
```

Le résultat renvoyé est 11, car la fonction connait la valeur de a. On dit que c'est une variable *globale*.

```
def loc():
        ll=2
        print(ll)

ll=ll+1
```

Ce programme affiche 2 puis `ll=ll+1 NameError: name 'll' is not defined` À l'intérieur de la fonction `loc()` la variable est utilisable, mais pas à l'extérieur. On dit que ll est *locale*.

III Alternatives : `if` ... `elif` ... `else`

III.1 Syntaxe

```
def signe(x):
        if x>=0 :
                return 1
        else:
                return -1
```

Cette fonction correspond à la fonction mathématique définie sur $\mathbb{R}$ par

$$signe(x) = \begin{cases} 1 & \text{si } x \text{ est positif ou nul} \\ -1 & \text{sinon} \end{cases}$$

On peut rajouter d'autres alternatives avec `elif` (pour else if) :

```
def signep(x):
        if x>=0 :
                return 1
        elif x==0:
                return 0
        else
                return -1
```

Exercice 2 (valeur absolue, maximum)**.**

1. Écrire une fonction `absolue(x)` qui calcule la valeur absolue d'un réel x.
2. Écrire une fonction `max2(a,b)` qui calcule le maximum de deux réels a et b.

III.2 Opérateurs logiques, comparaison

Un *booléen* est un type de variable qui ne peut prendre que deux valeurs True et False.

Comparaison	Exemple	Nom	Mathématiques
>	x > y	supériorité stricte	$>$
<	x < y	infériorité stricte	$<$
==	x == y	égalité	$=$
!=	x!= y	différence	$\neq$
>=	x >= y	supériorité large	$\geqslant$
<=	x <= y	infériorité large	$\leqslant$

```
>>>2>1
True

>>> 1!=0
True
>>> 'a'<'b'
True

>>> 'ab'<'ba'
True

>>> 'baaaaaaa'<'aaaa'
False
```

On peut aussi effectuer des opérations sur les booléens, ces opérations correspondent aux connecteurs logiques vus pendant le cours de mathématiques.

Opérateur	Utilisation	Sens
and	x and y	Et : vrai si les deux opérandes sont vraies
or	x or y	Ou : vrai si au moins une des opérandes est vraie
not	not x	Négation

```
>>> (3>2) and (2!=1)
True

>>> True and not False
True
```

Exercice 3 (maximum de trois nombre)**.**
Écrire une fonction `max3(a,b,c)` qui calcule le maximum de trois réels a, b et c.

IV Répétitions

IV.1 Énumérations : boucle for

IV.1.a Généralités

```
for i in range(1,6):
        print(i)
```

L'opération d'affichage est répétée pour tout les entiers i vérifiant $1 \leqslant i < 10$, on obtient

```
1
2
3
4
5
```

La fonction suivante calcule $\sum_{k=0}^{n} k^3$.

```
def sommecube(n):
        s=0
        for i in range(1,n+1):
                s=s+i**3

        return s
```

Exercice 4 (Factorielle)**.**
On cherche à créer une fonction factorielle, corrigez le programme suivant :

```
def facto(n)
        p=0
        for i in range(1,n):
                p=p*i

        return p
```

IV.1.b range

1. `range(a)` liste les entiers i tels que $0 \leqslant i < a$. a doit être positif sinon la boucle est vide.
2. `range(a,b)` liste les entiers i tels que $a \leqslant i < b$. On doit avoir $a < b$ sinon la boucle est vide.
3. `range(a,b,s)` liste les entiers a, $a+s$, $a+2s$, $a+3s$ $a+ks$ tels que $a+ks < b$ et $k \in \mathbb{N}$. s peut être négatif.

Exercice 5.
Écrire une fonction qui calcule $\prod_{k=1}^{n} 2k+1$.

IV.1.c Autres énumérables ⛰

On peut trouver des boucles for sans `range()` :

```
liste = ['Alain', 'Bernard', "Candice"]
for nom in liste:
        print("Bonjour", nom)
```

On obtient :

```
Bonjour Alain
Bonjour Bernard
Bonjour Candice
```

```
for i in "abcde":
        print(i)
```

On obtient :

```
a
b
c
d
e
```

IV.2 Répétitions conditionnelles : boucle `while`

L'itération précédente ne peut pas toujours être toujours utilisée, notamment lorsque l'on ne connaît pas le nombre de répétitions

```
def partieentiere(x):
        n=0
        while n<=x:
                n=n+1

        return(n-1)
```

Exercice 6.
En utilisant un `if` modifier la fonction précédente pour qu'elle calcule aussi la partie entière dans le cas où x est négatif.

La condition d'arrêt suit la même syntaxe pour un `if`.

Exercice 7.
On sait que $\lim_{n\to+\infty} \frac{1}{n!} = 0$, écrire un programme qui calcule le plus petit entier naturel tel que $\frac{1}{n!} < 10^{-10}$

On peut facilement créer des boucles infinies

```
i=1
while i<10:
print(i)
```

Exercice 8 (de for à while)**.**
Transformer les scripts suivants pour transformer les boucles `for` en boucle `while`.

1.

```
for i in range(1,10):
        print(i)
```

2.

```
s=0
for i in range(1,100,3):
        s=s+i
```

Exercice 9 (Parité)**.**
Écrire une fonction `paire(n)` qui pour un entier n renvoie `True` si n est pair et `False` sinon.
Deux lignes au maximum!

Exercice 10 (Puissance)**.**
Sans utiliser l'opérateur `**` écrire une fonction `puissance(x,n)` qui calcule x^n

- x est un réel et n un entier.
- L'entier n peut être négatif

Exercice 11 (Suite de Fibonacci)**.**
On défini la suite $(F_n)_{n\in\mathbb{N}}$ par

$$F_0 = F_1 = 1 \qquad \forall n \in \mathbb{N} \qquad F_{n+2} = F_{n+1} + F_n$$

Écrire une fonction `Fibonacci(n)` qui pour tout entier naturel n calcule F_n. Le résultat doit être valide pour $n = 0$ et $n = 1$.

Exercice 12 (triplet pythagoricien)**.**
$(a,b,c) \in (\mathbb{N}^*)^3$ forment un *triplet pythagoricien* si et seulement si

$$a^2 + b^2 = c^2 \qquad \text{et} \qquad a < b < c$$

1. Écrire une fonction `triplet(a,b,c)` qui vérifie si (a,b,c) forment un triplet.
2. On admet qu'il existe un unique triplet pythagoricien tels que $a+b+c = 1000$, écrire un programme qui calcule ces entiers a,b,c.

IV.3 Boucles imbriquées

On utilisera souvent une boucle à l'intérieur d'une autre boucle.

```
def sommedouble1(n):
        s=0
        for i in range(1,n+1):
                for j in range(1,n+1):
                        s=s+\dfrac{1}{i+j}
```

```
        return s
```

La fonction précédente calcule la valeur de $\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{1}{i+j}$

Exercice 13. 1. Écrire une fonction `sommedouble2(n,p):` qui calcule ma valeur de $\sum_{i=1}^{n}\sum_{j=1}^{p}\frac{i+j}{i*j}$

2. Écrire une fonction `sommedouble3(n):` qui calcule ma valeur de $\sum_{1\leqslant i\leqslant j\leqslant n}\frac{1}{i^2+j^2}$

V Méthodes générales sur les listes et les tableaux

V.1 Création d'une liste

La principale façon de créer une liste est de partir d'une liste vide et de rajouter des éléments à l'aide d'une boucle `for` ou `while` et la méthode `.append()`.

```
L=[]
for ....:
        L.append(.....)
```

V.2 Parcours séquentiel d'une liste, avec indice

Dans ce type d'algorithme, on doit parcourir l'ensemble de la liste, et on s'intéresse à l'indice des éléments. On utilise

```
for i in range(len(L)):
        ...#pour accéder à l'élément on utilise L[i]
```

Exemple :

- Recherche de l'indice de l'élément maximum, de l'élément minimum.
- Recherche du nombre d'occurrence d'un élément x dans la liste, de la liste des indices d'un éléments.
- Calcul d'une somme, d'un produit d'une moyenne.

V.3 Parcours séquentiel d'une liste, sans indice

Dans ce type d'algorithme, on doit parcourir l'ensemble de la liste, et on s'intéresse à la valeur des éléments. On utilise

```
for x in L:
        ...#pour acceder à l'élément on utilise x
```

Exemple :

- Recherche de la valeur de l'élément maximum, de l'élément minimum.
- Recherche du nombre d'occurrence d'un élément x dans la liste .
- Calcul d'une somme, d'un produit d'une moyenne

V.4 Parcours séquentiel, avec interruption

Dans ce type d'algorithme on parcourt la liste jusqu'à une certaine condition soit réalisée. On trouve souvent le terme "premier élément" dans l'énoncé.

Exemple :

- Première occurrence d'un élément x dans une liste L.
- Présence ou absence d'un élément x dans une liste L.
- La liste est elle triée dans l'ordre croissant?

```
def f()    :
        for i in range (len(L)):
                if     :
                        return   # Le return arrete la boucle
                        ↪  et la fonction
        return      # valeur si la Liste a été parcourue en
        ↪  entier
```

Avec un while et un drapeau ⛰

```
drapeau=False
i=0
while i<len(L) and drapeau=False:
        if ....    :
                drapeau =True
        i=i+1
```

VI Algorithmes à connaître sur les listes

VI.1 Maximum et minimum

VI.1.a Position

```
def max(L):
        ''' position du maximum, on suppose la liste non vide
        ↪  renvoie le plus petit entier i tel que L[i]=
        ↪  Max(L)'''
        resultat=0
        for i in range(len(L)):
                if L[i]>L[resultat]:
                        resultat=i
        return resultat
```

VI.1.b Valeur avec range

```
def max(L):
```

```
        ''' valeur du maximum, on suppose la liste non vide
        ↪  renvoie la valeur  de  Max(L)'''
        resultat=L[0]
        for i in range(len(L)):
                if L[i]>resultat:
                        resultat=L[i]
        return resultat
```

VI.1.c Valeur sans range

```
def max(L):
        ''' valeur du maximum, on suppose la liste non vide
        ↪  renvoie la valeur  de  Max(L)'''
        resultat=L[0]
        for x in L:
                if x>resultat:
                        resultat=x
        return resultat
```

VI.1.d Adaptations à savoir faire

En changeant l'inégalité en $<, \geqslant, \leqslant$ vous devez pouvoir écrire une fonction qui :

- Donne l'indice du maximum, et si il y a plusieurs maxima , renvoie l'indice le plus grand.
- Donne l'indice le plus grand ou le plus petit, ou la valeur du minimum.

VI.1.e Adaptations avancées ⛰

- Valeur et position du maximum.
- Valeur du maximum et nombre de fois où le maximum apparaît.
- Liste des positions des maxima.

VI.2 Deuxième maximum

```
def deuxiememax(L):
        ''' renvoie la valeur du deuxième maximum, c'est à
        ↪  dire la deuxième valeur la plus grande de L. On
        ↪  suppose la liste contient deux éléments et que
        ↪  tous les éléments sont distincts'''
        if L[0]>L[1]:
                max1=L[0]
                max2=L[1]
        else:
                max2=L[0]
                max1=L[1]

        for i in range(2,len(L)):
                if L[i]>max1:
                        max2=max1
                        max1=L[i]
                elif L[i]>max2:
                        max2=L[i]
        return max2
```

VI.2.a Adaptations à savoir faire

- Version sans range
- Deuxième minimum
- Position du deuxième minimum et du deuxième maximum

VI.3 Occurrences

Sont regroupés dans cette partie tous les algorithmes permettant de compter le nombre d'occurrence, de connaître la présence d'un élément dans une liste.

VI.3.a Fonctions déjà présentes dans python

- méthode **count()** compte le nombre de fois ou un élément apparaît dans une liste.

```
L=[1,2,3,2,1]
print(L.count(5))
print(L.count(2))
```

  Les valeurs affichées sont 0 et 2

- Opérateur **in** teste la présence ou l'absence d'un élément dans une liste, le résultat est un booléen.

```
L=[1,2,3,2,1]
print( 2 in L)
print( 'a' in L)
```

  Les valeurs affichées sont True et False

- (hors programme mais utile) la méthode **index** renvoie la première position d'un élément dans une liste, si l'élément est absent une exception *ValueError* est soulevée.

```
L=[1,2,3,2,1]
print(L.index(2))
```

  et

```
        L=[1,2,3,2,1]
        print(L.index('a'))
```

  renvoie le message d'erreur " print(L.index('a')) ValueError : 'a' is not in list"

Même si ces fonctions existent et que nous les utiliseront, il faut être capable d'écrire les fonctions suivantes de façon basique.

VI.3.b Première occurrence

Avec for

```
def occurrence(L,x):
        '''renvoie le plue petit indice i tel que L[i]=c, si
        ↪ l'élément est absent la valeur renvoyée est
        ↪ None'''
        for i in range(len(L)):
                if L[i]==x:
                        return i #interruption de la fonction
        return None
```

Avec While et drapeau ⛰

```
def occurrence(L,x):
        '''renvoie le plue petit indice i tel que L[i]=c, si
        ↪ l'élément est absent la valeur renvoyée est
        ↪ None'''
        drapeau=False
        i=0
        while drapeau==False and i<len(L):
                if L[i]== x:
                        drapeau=True
                i=i+1tcolorbox

        if i==len(L):#la liste a été parcourue en entier
                return None
        else :
                return i-1
```

Adaptations à savoir faire

- Utiliser un for sans range.
- changer la valeur renvoyée en cas d'absence en -1.

VI.3.c Présence d'un élément

For sans range

```
def presence(L,x):
        '''renvoie True si x est dans L, False sinon'''
        for e in L:
                if x==e:
                        return True
        return False
```

While ⛰

```
def presence(L,x):
        '''renvoie True si x est dans L, False sinon'''
        i=0
        drapeau=False
        while i<len(L) and drapeau== False:
                if L[i]== x:
                        drapeau=True
                i=i+1
        return drapeau
```

Adaptations à savoir faire

- version for avec range.

VI.3.d Nombre d'occurrence

Version sans range

```
def compte(L,x):
        '''compte le nombre d'occurrence de x dans L'''
        nb=0
        for e in L:
                if e==x:
                        nb=nb+1
        return nb
```

Version avec range

```
def compte(L,x):
        '''compte le nombre d'occurrence de x dans L'''
        nb=0
        for i in range(len(L)):
                if L[i]==x:
                        nb=nb+1
        return nb
```

Liste des occurrences ⛰

```
def liste_occurrence(L,x):
        '''renvoie la liste des indices i tels que L[i]=x'''
        R=[]
        for i in range(len(L)):
                if L[i]==x:
                        R.append(i)
        return R
```

VII Algorithmes plus avancés sur les listes, boucles imbriquées

VII.1 Recherche par dichotomie dans une liste triée

On suppose que la liste L est triée dans l'ordre croissant et on cherche la position de x dans L, on renvoie -1 si x n'est pas présent. On utilise un algorithme de dichotomie (voir le TP).

```
def dichotomie(L,x):
        '''retourne la position de x -1 si x est absent'''
        debut=0
        fin=len(L)-1#position du dernier élément
        while debut<fin:
                milieu=(debut+fin)//2 # quotient division
                ↪  euclidienne
                if L[milieu]==x:
                        return milieu
                elif L[milieu]>x:
                        fin=milieu
                else:
                        debut=milieu+1
        if L[debut]==x:
                return debut
        else:
                return -1
```

Pour tester si la liste est bien triée

```
        def esttriee(L):
        '''teste si la liste est triée dans l'ordre
        ↪  croissant'''
        for i in range(1,len(L)):#attention au départ
                if L[i-1]>L[i]:
                        return False
        return True
```

VII.1.a Adaptation à savoir faire

- Renvoie juste True/False.
- Cas où la liste est triée dans l'ordre décroissant.
- Tester si la liste est bien triée dans l'ordre décroissant.

VII.2 Trouver les deux éléments les plus éloignés ⛰

VII.2.a Valeur

On dispose d'une liste d'au moins deux éléments on cherche à calculer la valeur maximale de |L[i]-L[j]| pour i j deux indices

```
def eloignes(L):
        ''' L a au moins deux éléments, calcule la valeur
        ↪  maximale  de |L[i]-L[j]| pour i j deux indices
        ↪  '''
        resultat=abs(L[0]-L[1])
        for i in range(len(L)):
                for j in range(i+1,len(L)):
                        if abs(L[i]-L[j])>resultat:
                                resultat=abs(L[i]-L[j])
        return resultat
```

VII.2.b Positions

```
def eloignes(L):
        ''' L a au moins deux éléments, calcule la position
        ↪  du couple (k,l) qui realise la  valeur  maximale
        ↪  de |L[i]-L[j]| par construction k<l et  sont
        ↪  minimaux'''
        k,l=0,1 #initialisation
        distance=abs(L[0]-L[1])
        for i in range(len(L)):
                for j in range(i+1,len(L)):
                if abs(L[i]-L[j])>distance:
                        distance=abs(L[i]-L[j])
                        k,l=i,j
        return k,l
```

VII.2.c Adaptations ⛰

- Renvoyer les valeurs des éléments les plus éloignés
- Renvoyer le couple des positions des éléments les plus proches qui ne sont pas égaux.

VII.3 Recherche d'un motif ⛰

On cherche à savoir si une sous liste (ou motif apparaît dans liste) Par exemple si L=[1,2,3,4,5] les motifs [1,2] et [4,5] sont dans L mais pas [5,4] et [1,1,2]

```
def motif(liste,motif):
        '''détermine si motif est une sous-liste de liste'''
        L=len(liste)# longueur de la liste
        l=len(motif)#longueur du motif
        drapeau=False
        i=0
        j=0
        while i+l <=L and drapeau ==False:
                j=0
                drapeau2=True
                while j<l and drapeau2==True:
```

```
                    drapeau2=(liste[i+j]==motif[j])
                    j=j+1
                drapeau=drapeau2
                i=i+1
        return drapeau
```

VII.3.a Adaptations

- Remplacer les boucles while par des boucles for.
- Nombre d'occurrence du motif.
- Position du motif.
- Liste des positions de motif dans liste.

VIII Autres algorithmes

VIII.1 Suite définies par récurrence

On veut calculer les termes successifs d'une suite définie par :

$$u_0 \in \mathbb{R}, \qquad \forall n \in \mathbb{N} \quad u_{n+1} = u_n$$

On ne se préoccupe pas de l'ensemble de définition de f. Voici un exemple :

$$u_0 \in \mathbb{R}, \qquad \forall n \in \mathbb{N} \quad u_{n+1} = u_n^2 - 1$$

```
def f(x):
        return x**2-1

def suite(u0,n):
        u=u0
        for i in range(n):
                u=f(u)
        return u
```

Adaptations à savoir faire

- Remplacer la boucle for par une boucle while, avec une condition portant sur la valeur de u_n.
- Sauvegarder toutes les valeurs calculées dans une liste.
- calculer le somme des termes de la suite (voir partie suivante).

Adaptations avancées ⛰

- La fonction f devient un argument
- Tracer le graphe de la fonction et de la suite avec "l'escargot".

VIII.2 Calcul de somme, de produit

VIII.2.a Somme et produit simple

Structure

```
def somme(....):
        S=0
        for    in    :
                S=S+....
        return S
```

```
def produit(....):
        P=1
        for    in    :
                P=P*....
        return P
```

Il faut bien faire attention au décalage dans le range.

Exemples Pour calculer

$$\sum_{i=1}^{n} i^3$$

```
def somme(n):
        S=0
        for i in range(n+1):
                S=S+i**3
        return S        -
```

Pour calculer

$$\prod_{i=1}^{n} \frac{1}{i^2+1}$$

```
def produit(n):
        P=1
        for i in range(1,n+1):
                P=P*1/(i**2+1)
        return P
```

VIII.2.b Utilisation de boucles imbriquées

Pour calculer des sommes doubles, on utilise

```
def somme(....  ):
        S=0
        for i in range.....  :
                for j in range.....  :
                        S=S+
```

```
        return S
```

$$S_1 = \sum_{1 \leqslant i,j \leqslant n} ij \qquad S_2 = \sum_{1 \leqslant i \leqslant j \leqslant n} i^j \qquad S_3 = \sum_{\substack{1 \leqslant i \leqslant n \\ 1 \leqslant j \leqslant p}} (i+j)$$

```
def S1(n):
        S=0
        for i in range(n+1):
                for j in range(n+1):
                        S=S+i*j
        return S

def S2(n):
        S=0
        for i in range(n+1):
                for j in range(i+1,n+1):
                        S=S+i**j
        return S

def S3(n,p):
        S=0
        for i in range(n+1):
                for j in range(p+1):
                        S=S+i+j
        return S
```

Glossaire

Affectation Action de donner une valeur à une variable

Bloc Portion du code d'un programme qui sont exécutées séquentiellement. En python les blocs sont marqués par la présence d'indentations régulières. Ils forment par exemple l'intérieur des boucles, des fonctions

Boucles instruction `for, while...` qui permettent de répéter des instructions.

Branchement/ instruction conditionnelle permet d'effectuer différentes actions en fonction de l'évaluation d'une condition

Éxécution ou interprétation Faire réaliser les instructions contenues dans un script.

Expression combinaison de constantes, variables et opérateurs qui respecte la bonne syntaxe. Exemple `5+2`

Fonction Suite d'opérations effectuant un calcul, une action, regroupées pour pouvoir être utilisées facilement de façon répétée. Une fonction comporte **un nom** et éventuellement des **arguments** (correspondant aux variables mathématiques) et une ou des valeurs renvoyées. L'utilisation ou **appel** de la fonction se fait en utilisant le nom.

Indentation décalage vers la droite. La touche utilisée est (tabulation)

Instruction Partie du code conduisant à une action ou une commande

Module Ensemble de fonctions supplémentaires, regroupées par thèmes **math** pour les fonctions mathéùmatiques `numpy` pour le calcul numérique. Pour *importer* un module, il faut utiliser l'une des différentes versions de la commande `import`

Opérateurs commande simple qui représente un calcul élémentaire : addition, multiplication...

Shell ou mode interactif permet d'exécuter des commandes simples, ou des opérations. Idéal pour tester en mode interactif les fonctions que vous avez créées

Script ou programme Suite d'instructions que l'on peut sauvegarder,exécuter...

Syntaxe Règles à suivre pour agencer correctement des opérations, des instructions. Comme dans un langage courant un programme qui ne respecte pas les règles de syntaxe ne pourra pas être interprété, mais en Python les règles de syntaxe sont beaucoup plus rigoureuses et moins flexibles.

Variable Nom qui fait référence à une valeur