

# Programming from examples

**Germain Vallverdu**

*Université de Pau et des Pays de l'Adour, IPREM - ECP CNRS UMR 5254  
Hélioparc Pau-Pyrénées, 2 av. du Président Angot  
64053 Pau cedex 9, France*

germain.vallverdu@univ-pau.fr





---

# Table des matières

---

<b>1</b>	<b>Énoncé des exercices</b>	<b>5</b>
1.1	Lire – écrire – calculer . . . . .	5
1.2	Faire une condition . . . . .	5
1.3	Répéter une action . . . . .	6
1.4	Variable indicée . . . . .	6
1.5	Sous programmes . . . . .	6
1.6	Premiers petits programmes . . . . .	6
<b>2</b>	<b>Algorithme – AlgoBox</b>	<b>9</b>
2.1	Lire – écrire – compter . . . . .	9
2.2	Faire une condition . . . . .	11
2.3	Répéter une action . . . . .	13
2.4	Variable indicée . . . . .	14
2.5	Sous programme . . . . .	15
2.6	Premiers petits programmes . . . . .	16
<b>3</b>	<b>Programmes en Fortran</b>	<b>27</b>
3.1	Lire – écrire – compter . . . . .	27
3.2	Faire une condition . . . . .	30
3.3	Répéter une action . . . . .	32
3.4	Variable indicée . . . . .	34
3.4.1	Déclaration statique des tableaux . . . . .	34
3.4.2	Déclaration dynamique des tableaux . . . . .	36
3.5	Sous programme . . . . .	38
3.6	Premiers petits programmes . . . . .	39

<b>4 Programmes en Python</b>	<b>49</b>
4.1 Lire – écrire – compter . . . . .	50
4.1.1 Versions sans fonction . . . . .	50
4.1.2 Versions avec une fonction . . . . .	51
4.2 Faire une condition . . . . .	53
4.2.1 Versions sans fonction . . . . .	53
4.2.2 Versions avec une fonction . . . . .	56
4.3 Répéter une action . . . . .	58
4.4 Variable indicée . . . . .	60
4.5 Sous programme . . . . .	62
4.6 Premiers petits programmes . . . . .	63

---

## Énoncé des exercices

---

Ce chapitre introduit les exemples qui seront présentés par la suite sous forme d’algorithmes et dans les différents langages de programmation. Les énoncés ci-dessous peuvent être vue comme les énoncés d’exercices dont l’objectif serait d’écrire les programmes donnés dans les chapitres suivants.

### 1.1 Lire – écrire – calculer

1. Écrire un programme qui demande à l’utilisateur la longueur et la largeur d’un rectangle et calcule l’aire.
2. Écrire un programme qui calcule le reste de la division de deux entiers demandés à l’utilisateur.
3. Écrire un programme qui calcule le périmètre d’un cercle après avoir demandé son rayon.

### 1.2 Faire une condition

1. Écrire un programme qui demande deux chiffres à l’utilisateur et affiche le plus grand.
2. Écrire un programme qui calcule la racine carré d’un nombre demandé à l’utilisateur.
3. Écrire un programme qui calcule les racines d’un polynôme du second degré.

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

$$\Delta = b^2 - 4ac$$

$$\begin{cases} \Delta > 0 & x = \frac{-b \pm \sqrt{\Delta}}{2a} & 2 \text{ solutions réelles} \\ \Delta = 0 & x = \frac{-b}{2a} & \text{une unique solution} \\ \Delta < 0 & x = \frac{-b \pm i\sqrt{|\Delta|}}{2a} & 2 \text{ solutions complexes} \end{cases}$$

### 1.3 Répéter une action

Les deux exercices suivants consistent à calculer les valeurs d'une fonction de votre choix dans l'intervalle  $[-5; 5]$  avec

1. un pas de 1.
2. avec un pas de 0.5.

### 1.4 Variable indicée

Les deux exercices suivants ont pour objectif d'introduire l'utilisation des variables indicées ou tableaux.

1. Écrire un programme qui crée une liste contenant les premiers entiers impairs et affiche cette liste.
2. Écrire un programme qui crée une liste contenant une série de nombres pseudo-aléatoires et affiche cette liste.

### 1.5 Sous programmes

Reprendre l'exercice consistant à calculer les valeurs d'une fonction dans l'intervalle  $[-5; 5]$  en créant une fonction qui sera appelée pour calculer les valeurs souhaitées.

Les autres exercices pourront être retravailler avec pour objectif de rendre fonctionnel les actions qu'ils réalisent.

### 1.6 Premiers petits programmes

Les exemples suivants constituent des cas plus concrets pouvant nécessiter d'associer l'écriture d'une condition et d'une boucle. Ils sont classés par ordre de complexité croissante.

1. Écrire un programme qui calcule la factorielle d'un entier naturel.

Rappel :

$$\begin{cases} n! = 1 \times 2 \times 3 \times \dots \times n & \forall n \in \mathbb{N}^* \\ n! = 1 & n = 0 \end{cases}$$

2. Écrire un programme qui calcule le produit des  $n$  premiers entiers impairs.
3. Écrire un programme qui calcule la somme des  $n$  premiers entiers naturels.
4. Une population décroît de 40% tous les 3 ans. La population étant considérée négligeable lorsqu'elle est inférieure à 0.1% de sa valeur initiale, au bout de combien d'année l'extinction est-elle atteinte ?

5. Écrire un programme qui construit une liste de nombres compris entre 0 et 100 puis cherche le minimum et le maximum dans cette liste.
6. La marche aléatoire d'un point peut être modélisée de la façon suivante :

$$\vec{r}(t + dt) = \vec{r}(t) + a\hat{\vec{R}}$$

où  $a$  désigne l'amplitude du déplacement aléatoire et  $\hat{\vec{R}}$  est un vecteur aléatoire dont les composantes sont comprises entre -1 et 1.

Écrire un programme qui met en œuvre une marche aléatoire dans un espace à deux dimensions et affiche à chaque pas de temps les coordonnées du point.

7. Le nombre  $\pi$  peut être calculé par un processus dit de Monte Carlo, mettant en oeuvre le tirage de nombres aléatoires. Le principe est le suivant : La probabilité qu'un point, de coordonnées  $(x, y)$ ,  $x \in [0, 1]$  et  $y \in [0, 1]$ , choisi aléatoirement, soit dans un cercle de rayon 1 est égale au rapport de l'aire du quart de cercle de rayon 1 compris dans le carré de largeur 1 et l'aire de ce carré. Écrire un programme qui calcule le nombre  $\pi$  par cette méthode.
8. Écrire un programme qui calcule l'intégrale d'une fonction par la méthode des trapèze.
9. Écrire un programme qui calcule l'intégrale d'une fonction par la méthode de Simpson.
10. Écrire un programme qui met en œuvre le procédé d'orthogonalisation de Gramm-Schmidt dont voici une brève description : Soit  $\vec{u}$  et  $\vec{v}$  deux vecteurs quelconques, le vecteur  $\vec{v}'$  orthogonal au vecteur  $\vec{u}$  est obtenu par :

$$\vec{v}' = \vec{v} - \frac{(\vec{u} \cdot \vec{v})}{\|\vec{u}\|^2} \vec{u}$$





## Algorithme – AlgoBox

---

### 2.1 Lire – écrire – compter

Algorithme 2.1: Calcul de l'aire d'un rectangle.

```
1 // aire_rectangle - 11.03.2014
2
3 // *****
4 // Calcul de l'aire d'un rectangle
5 // *****
6
7 VARIABLES
8   largeur EST_DU_TYPE NOMBRE
9   longueur EST_DU_TYPE NOMBRE
10 DEBUT_ALGORITHME
11   // Lecture des variables
12   LIRE largeur
13   LIRE longueur
14   //Affiche le résultat
15   AFFICHER "Aire du rectangle : "
16   AFFICHERCALCUL largeur * longueur
17 FIN_ALGORITHME
```

Algorithme 2.2: Calcul du reste de la division de deux entiers.

```

1 // reste - 11.03.2014
2
3 // *****
4 // Calcul du reste de la division de deux entiers
5 // *****
6
7 VARIABLES
8   dividende EST_DU_TYPE NOMBRE
9   diviseur EST_DU_TYPE NOMBRE
10  reste EST_DU_TYPE NOMBRE
11 DEBUT_ALGORITHME
12   //Lecture des valeurs
13   LIRE dividende
14   AFFICHER "dividende = "
15   AFFICHER dividende
16   LIRE diviseur
17   AFFICHER "diviseur = "
18   AFFICHER diviseur
19   //Calcul du reste
20   reste PREND_LA_VALEUR dividende % diviseur
21   //Affichage du resultat
22   AFFICHER "reste = "
23   AFFICHER reste
24 FIN_ALGORITHME

```

Algorithme 2.3: Calcul du périmètre d'un cercle.

```

1 // perimetre - 11.03.2014
2
3 // *****
4 // Calcul du périmètre d'un cercle
5 // *****
6
7 VARIABLES
8   rayon EST_DU_TYPE NOMBRE
9   perimetre EST_DU_TYPE NOMBRE
10 DEBUT_ALGORITHME
11   // lecture du rayon
12   LIRE rayon
13   AFFICHER "rayon = "
14   AFFICHER rayon
15   // calcul du périmètre
16   perimetre PREND_LA_VALEUR 2 * Math.PI * rayon
17   // affichage du résultat
18   AFFICHER "périmètre = "
19   AFFICHER perimetre
20 FIN_ALGORITHME

```

## 2.2 Faire une condition

Algorithme 2.4: Affiche le nombre le plus grand.

```
1 // le_plus_grand - 11.03.2014
2
3 // *****
4 // Lire x et y et dire lequel est le plus grand
5 // *****
6
7 VARIABLES
8   x EST_DU_TYPE NOMBRE
9   y EST_DU_TYPE NOMBRE
10 DEBUT_ALGORITHME
11   //lecture de x et y
12   LIRE x
13   LIRE y
14   AFFICHER "x = "
15   AFFICHER x
16   AFFICHER "y = "
17   AFFICHER y
18   //test entre x et y
19   SI (x > y) ALORS
20     DEBUT_SI
21     AFFICHER "x est plus grand"
22     FIN_SI
23   SINON
24     DEBUT_SINON
25     SI (y > x) ALORS
26       DEBUT_SI
27       AFFICHER "y est plus grand"
28       FIN_SI
29     SINON
30       DEBUT_SINON
31       AFFICHER "x et y sont égaux"
32       FIN_SINON
33     FIN_SINON
34 FIN_ALGORITHME
```

Algorithme 2.5: Calcul des racines d'un polynôme de degré 2.

```

1  // discriminant - 11.03.2014
2
3  // *****
4  // Calcul des racines réelles d'un polynome de degré 2
5  // *****
6
7  VARIABLES
8      a EST_DU_TYPE NOMBRE
9      b EST_DU_TYPE NOMBRE
10     c EST_DU_TYPE NOMBRE
11     delta EST_DU_TYPE NOMBRE
12 DEBUT_ALGORITHME
13     AFFICHER "On va résoudre l'équation  $a \cdot x^2 + b \cdot x + c = 0$ "
14     //Lecture des variables
15     LIRE a
16     LIRE b
17     LIRE c
18     //Calcul du discriminant
19     delta PREND_LA_VALEUR  $b^2 - 4 \cdot a \cdot c$ 
20     AFFICHER "Delta = "
21     AFFICHER delta
22     //Test du discriminant
23     SI (delta >= 0) ALORS
24         DEBUT_SI
25         SI (delta == 0) ALORS
26             DEBUT_SI
27                 AFFICHER "L'équation a une seule solution"
28                 AFFICHERCALCUL  $-b / (2 * a)$ 
29             FIN_SI
30         SINON
31             DEBUT_SINON
32                 AFFICHER "L'équation a deux solutions réelles"
33                 AFFICHERCALCUL  $(-b + \text{sqrt}(\text{delta})) / (2 * a)$ 
34                 AFFICHERCALCUL  $(-b - \text{sqrt}(\text{delta})) / (2 * a)$ 
35             FIN_SINON
36         FIN_SI
37     SINON
38         DEBUT_SINON
39             AFFICHER "L'équation a deux solutions complexes"
40         FIN_SINON
41 FIN_ALGORITHME

```

Algorithme 2.6: Calcul de la racine carré d'un nombre.

```

1  // racine_x - 11.03.2014
2
3  // *****
4  // Calcul de la racine carré de x si x est positif.
5  // *****
6
7  VARIABLES
8      x EST_DU_TYPE NOMBRE
9  DEBUT_ALGORITHME
10     // Lecture de x
11     LIRE x
12     AFFICHER "x = "
13     AFFICHER x
14     // test de la valeur de x et calcul de sqrt(x)
15     SI ( x >= 0) ALORS
16         DEBUT_SI
17             AFFICHER "RACINE(x) = "
18             AFFICHERCALCUL sqrt(x)
19         FIN_SI
20     SINON
21         DEBUT_SINON
22             AFFICHER "x est négatif"
23         FIN_SINON
24 FIN_ALGORITHME

```

## 2.3 Répéter une action

Algorithme 2.7: Illustration de l'utilisation d'une boucle POUR

```

1  calc_val_fonction_1 - 11.03.2014
2
3  *****
4  calcule des valeurs de la fonction  $f(x) = x^2$  pour x dans [-5 ; 5]
5  *****
6
7  VARIABLES
8      x EST_DU_TYPE NOMBRE
9  DEBUT_ALGORITHME
10     //boucle sur les valeurs de x
11     POUR x ALLANT_DE -5 A 5
12         DEBUT_POUR
13             AFFICHERCALCUL x * x
14         FIN_POUR
15 FIN_ALGORITHME

```

Algorithme 2.8: Illustration de l'utilisation d'une boucle TANT\_QUE.

```

1 // calc_val_fonction_2 - 11.03.2014
2
3 // *****
4 // calcule des valeurs de la fonction  $f(x) = x^2$  pour  $x$  dans  $[-5 ; 5]$ 
5 // *****
6
7 VARIABLES
8   x EST_DU_TYPE NOMBRE
9   pas EST_DU_TYPE NOMBRE
10 DEBUT_ALGORITHME
11   //initialisation
12   x PREND_LA_VALEUR -5
13   pas PREND_LA_VALEUR 0.5
14   //boucle sur les valeurs de x
15   TANT_QUE (x <= 5) FAIRE
16       DEBUT_TANT_QUE
17       AFFICHERCALCUL x * x
18       x PREND_LA_VALEUR x + pas
19       FIN_TANT_QUE
20 FIN_ALGORITHME

```

## 2.4 Variable indicée

Algorithme 2.9: Création d'une liste contenant les premiers entiers impairs.

```

1 // liste_impair - 11.03.2014
2
3 // *****
4 // Remplissage et affichage d'une liste des 20 premiers entiers impairs
5 // *****
6
7 VARIABLES
8   x EST_DU_TYPE LISTE
9   i EST_DU_TYPE NOMBRE
10  n EST_DU_TYPE NOMBRE
11 DEBUT_ALGORITHME
12   //Nombre d'entier
13   n PREND_LA_VALEUR 20
14   //Remplissage de la liste
15   POUR i ALLANT_DE 0 A n - 1
16       DEBUT_POUR
17       x[i] PREND_LA_VALEUR 2 * i + 1
18       FIN_POUR
19   //Affichage de la liste
20   AFFICHER "Liste des 20 premiers nombres impaires"
21   POUR i ALLANT_DE 0 A n - 1
22       DEBUT_POUR
23       AFFICHER x[i]
24       FIN_POUR
25 FIN_ALGORITHME

```

Algorithme 2.10: Création d'une liste de nombres pseudo-aléatoires.

```

1 // liste_random - 11.03.2014
2
3 // *****
4 // Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
5 // *****
6
7 VARIABLES
8   i EST_DU_TYPE NOMBRE
9   n EST_DU_TYPE NOMBRE
10  maListe EST_DU_TYPE LISTE
11  x EST_DU_TYPE NOMBRE
12 DEBUT_ALGORITHME
13   //Choix du nombre de valeurs
14   n PREND_LA_VALEUR 15
15   //Remplissage et affichage de la liste
16   POUR i ALLANT_DE 1 A n
17     DEBUT_POUR
18     x PREND_LA_VALEUR 2. * random() - 1.
19     maListe[i] PREND_LA_VALEUR x
20     AFFICHER x
21   FIN_POUR
22 FIN_ALGORITHME

```

## 2.5 Sous programme

Algorithme 2.11: Utilisation de l'écriture d'une fonction et de l'appel à cette fonction.

```

1 // calc_val_fonction_3 - 11.03.2014
2
3 // *****
4 // calcule des valeurs de la fonction  $f(x) = x^2$  pour x dans [-5 ; 5]
5 // *****
6
7 VARIABLES
8   x EST_DU_TYPE NOMBRE
9   pas EST_DU_TYPE NOMBRE
10 DEBUT_ALGORITHME
11   //initialisation
12   x PREND_LA_VALEUR -5
13   pas PREND_LA_VALEUR 0.5
14   //boucle sur les valeurs de x
15   TANT_QUE (x <= 5) FAIRE
16     DEBUT_TANT_QUE
17     AFFICHERCALCUL F1(x)
18     x PREND_LA_VALEUR x + pas
19   FIN_TANT_QUE
20 FIN_ALGORITHME
21
22 // Fonction numérique utilisée :
23 F1(x)=x*x

```

## 2.6 Premiers petits programmes

Algorithme 2.12: Calcul de la factorielle d'un nombre entier.

```

1 // factorielle - 11.03.2014
2
3 // *****
4 // Calcul de factorielle n
5 // *****
6
7 VARIABLES
8   i EST_DU_TYPE NOMBRE
9   n EST_DU_TYPE NOMBRE
10  factorielle EST_DU_TYPE NOMBRE
11 DEBUT_ALGORITHME
12   //lecture de n
13   LIRE n
14   AFFICHER "Calcul de factorielle "
15   AFFICHER n
16   //initialisation
17   factorielle PREND_LA_VALEUR 1
18   // calcul de la factorielle
19   POUR i ALLANT_DE 2 A n
20     DEBUT_POUR
21     factorielle PREND_LA_VALEUR factorielle * i
22     FIN_POUR
23   //affichage du résultat
24   AFFICHER "Résultat = "
25   AFFICHER factorielle
26 FIN_ALGORITHME

```



Algorithme 2.13: Calcul d'un produit.

```
1 // produit - 11.03.2014
2
3 // *****
4 // Calcul du produit des n premiers entiers impairs
5 // *****
6
7 VARIABLES
8   i EST_DU_TYPE NOMBRE
9   n EST_DU_TYPE NOMBRE
10  produit EST_DU_TYPE NOMBRE
11 DEBUT_ALGORITHME
12   //lecture de n
13   LIRE n
14   AFFICHER "Calcul du produit des entiers impairs entre 1 et "
15   AFFICHER n
16   //initialisation
17   produit PREND_LA_VALEUR 1
18   POUR i ALLANT_DE 3 A n
19     DEBUT_POUR
20     SI (i % 2 == 1) ALORS
21       DEBUT_SI
22       produit PREND_LA_VALEUR produit * i
23       FIN_SI
24     FIN_POUR
25   //affichage du résultat
26   AFFICHER "Résultat = "
27   AFFICHER produit
28 FIN_ALGORITHME
```

## Algorithme 2.14: Calcul d'une somme.

```
1 // somme - 11.03.2014
2
3 // *****
4 // Calcul de la somme des n premiers entiers
5 // *****
6
7 VARIABLES
8   n EST_DU_TYPE NOMBRE
9   i EST_DU_TYPE NOMBRE
10  somme EST_DU_TYPE NOMBRE
11 DEBUT_ALGORITHME
12   //Lecture de n
13   LIRE n
14   AFFICHER "Calcul de la somme des entiers de 1 à "
15   AFFICHER n
16   //initialisation de la somme
17   somme PREND_LA_VALEUR 0
18   POUR i ALLANT_DE 1 A n
19     DEBUT_POUR
20     somme PREND_LA_VALEUR somme + i
21     FIN_POUR
22   //affichage du résultat
23   AFFICHER "Résultat = "
24   AFFICHER somme
25 FIN_ALGORITHME
```

Algorithme 2.15: Calcul itératif de la décroissance d'une population.

```

1  // population - 11.03.2014
2
3  // *****
4  // Une population est réduite de 40% tous les 3 ans.
5  // Au bout de combien d'années la population est négligeable
6  // (inférieure à 0.1% de la population initiale) ?
7  // *****
8
9  VARIABLES
10  population EST_DU_TYPE NOMBRE
11  an EST_DU_TYPE NOMBRE
12  perte EST_DU_TYPE NOMBRE
13  seuil EST_DU_TYPE NOMBRE
14  DEBUT_ALGORITHME
15  //initialisation
16  population PREND_LA_VALEUR 100
17  an PREND_LA_VALEUR 0
18  perte PREND_LA_VALEUR 0.4
19  seuil PREND_LA_VALEUR 0.1 / 100 * population
20  //boucle sur les années
21  TANT_QUE (population > seuil) FAIRE
22  DEBUT_TANT_QUE
23  an PREND_LA_VALEUR an + 3
24  population PREND_LA_VALEUR population * (1 - perte)
25  AFFICHER an
26  AFFICHER " "
27  AFFICHER population
28  FIN_TANT_QUE
29  //résultats
30  AFFICHER "Au bout de "
31  AFFICHER an
32  AFFICHER " ans, la population est inférieure à 0.1% de la population initiale."
33  FIN_ALGORITHME

```

Algorithme 2.16: Recherche du maximum et du minimum dans une liste.

```

1  // minmax - 11.03.2014
2
3  // *****
4  // Recherche du maximum et du minimum dans une liste pseudo-aléatoire
5  // *****
6
7  VARIABLES
8      i EST_DU_TYPE NOMBRE
9      n EST_DU_TYPE NOMBRE
10     maListe EST_DU_TYPE LISTE
11     mini EST_DU_TYPE NOMBRE
12     maxi EST_DU_TYPE NOMBRE
13 DEBUT_ALGORITHME
14     //nombre de points
15     LIRE n
16     AFFICHER "n = "
17     AFFICHER n
18     //Remplissage d'une liste pseudo-aléatoire de nombres entre 0 et 100
19     POUR i ALLANT_DE 1 A n
20         DEBUT_POUR
21             maListe[i] PREND_LA_VALEUR 100. * random()
22             AFFICHER maListe[i]
23         FIN_POUR
24     //Recherche du minimum et du maximum
25     //initialisation
26     mini PREND_LA_VALEUR maListe[1]
27     maxi PREND_LA_VALEUR maListe[1]
28     POUR i ALLANT_DE 1 A n
29         DEBUT_POUR
30             //Recherche du minimum
31             SI (maListe[i] < mini) ALORS
32                 DEBUT_SI
33                     mini PREND_LA_VALEUR maListe[i]
34                 FIN_SI
35             //Recherche maximum
36             SI (maListe[i] > maxi) ALORS
37                 DEBUT_SI
38                     maxi PREND_LA_VALEUR maListe[i]
39                 FIN_SI
40         FIN_POUR
41     AFFICHER "maximum = "
42     AFFICHER maxi
43     AFFICHER "minimum = "
44     AFFICHER mini
45 FIN_ALGORITHME

```

Algorithme 2.17: Marche aléatoire d'un point dans un plan 2D.

```

1  // marche_aleatoire - 11.03.2014
2
3  // *****
4  // Marche aléatoire dans une plan 2D
5  // *****
6
7  VARIABLES
8      i EST_DU_TYPE NOMBRE
9      npas EST_DU_TYPE NOMBRE
10     amplitude EST_DU_TYPE NOMBRE
11     x EST_DU_TYPE LISTE
12     y EST_DU_TYPE LISTE
13     wx EST_DU_TYPE NOMBRE
14     wy EST_DU_TYPE NOMBRE
15 DEBUT_ALGORITHME
16     //paramètres de la marche
17     LIRE npas
18     LIRE amplitude
19     //initialisation
20     x[1] PREND_LA_VALEUR 0
21     y[1] PREND_LA_VALEUR 0
22     //marche aléatoire
23     POUR i ALLANT_DE 2 A npas
24         DEBUT_POUR
25             wx PREND_LA_VALEUR 2. * random() - 1.
26             wy PREND_LA_VALEUR 2. * random() - 1.
27             x[i] PREND_LA_VALEUR x[i - 1] + amplitude * wx
28             y[i] PREND_LA_VALEUR y[i - 1] + amplitude * wy
29         FIN_POUR
30     //représentation de la marche
31     POUR i ALLANT_DE 2 A npas
32         DEBUT_POUR
33             TRACER_SEGMENT (x[i - 1],y[i - 1])->(x[i],y[i])
34         FIN_POUR
35 FIN_ALGORITHME

```

Algorithme 2.18: Calcul du nombre  $\pi$  par la méthode Monte Carlo.

```

1  // pi - 14.03.2014
2
3  // *****
4  // Calcul du nombre pi par Monte Carlo.
5  // Principe : La probabilité qu'un point de coordonnées (x,y)
6  // avec x et y dans l'intervalle [0;1] soit dans le quart de
7  // cercle de centre (0,0) et de rayon 1 est égal au rapport des
8  // aires du quart de cercle de rayon 1 et du carré de largeur 1
9  // soit pi/4.
10 *****
11
12 VARIABLES
13   x EST_DU_TYPE NOMBRE
14   y EST_DU_TYPE NOMBRE
15   n EST_DU_TYPE NOMBRE
16   ntirage EST_DU_TYPE NOMBRE
17   i EST_DU_TYPE NOMBRE
18 DEBUT_ALGORITHME
19   //Lecture du nombre de tirage
20   LIRE ntirage
21   //initialisation
22   n PREND_LA_VALEUR 0
23   //calcul du nombre pi
24   POUR i ALLANT_DE 1 A ntirage
25     DEBUT_POUR
26       //tirage d'un point dans le carré de coté 1
27       x PREND_LA_VALEUR random()
28       y PREND_LA_VALEUR random()
29       SI (x*x + y*y < 1) ALORS
30         DEBUT_SI
31           n PREND_LA_VALEUR n + 1
32           TRACER_POINT (x,y)
33         FIN_SI
34       FIN_POUR
35   //Affichage des résultats
36   AFFICHER "pi = "
37   AFFICHERCALCUL 4.0 * n / ntirage
38   AFFICHER "% d'erreur = "
39   AFFICHERCALCUL (Math.PI - 4.0 * n / ntirage) / Math.PI * 100.
40 FIN_ALGORITHME

```

## Algorithme 2.19: Intégration par la méthode des trapèzes.

```

1  // trapeze - 11.03.2014
2
3  // *****
4  // Intégration par la méthode des trapèzes. L'intégration analytique
5  // de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
6  // application numérique -2.525369
7  // *****
8
9  VARIABLES
10     a EST_DU_TYPE NOMBRE
11     b EST_DU_TYPE NOMBRE
12     i EST_DU_TYPE NOMBRE
13     npas EST_DU_TYPE NOMBRE
14     integrale EST_DU_TYPE NOMBRE
15     analytique EST_DU_TYPE NOMBRE
16     pas EST_DU_TYPE NOMBRE
17     x EST_DU_TYPE NOMBRE
18 DEBUT_ALGORITHME
19     //valeur exacte
20     analytique PREND_LA_VALEUR -2.525369
21     //Lecture de l'intervalle
22     LIRE a
23     LIRE b
24     //Nombre de segments
25     LIRE npas
26     //initialisation
27     integrale PREND_LA_VALEUR 0
28     pas PREND_LA_VALEUR (b - a) / npas
29     //calcul de l'integrale
30     POUR i ALLANT_DE 1 A npas
31         DEBUT_POUR
32             x PREND_LA_VALEUR a + (i - 1) * pas
33             integrale PREND_LA_VALEUR integrale + pas * (F1(x) + F1(x + pas)) / 2
34         FIN_POUR
35     //résultat
36     AFFICHER "Résultat = "
37     AFFICHER integrale
38     AFFICHER "Résidu = "
39     AFFICHERCALCUL analytique - integrale
40     AFFICHER "Précision = "
41     AFFICHERCALCUL (analytique - integrale) / analytique * 100
42 FIN_ALGORITHME
43
44 // Fonction numérique utilisée :
45 F1(x)=(x * x - 3 * x - 6) * exp(-x)

```

## Algorithme 2.20: Intégration par la méthode de Simpson.

```

1  // simpson - 11.03.2014
2
3  // *****
4  // Intégration par la méthode de Simpson. L'intégration analytique
5  // de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
6  // application numérique -2.525369
7  // *****
8
9  VARIABLES
10     a EST_DU_TYPE NOMBRE
11     b EST_DU_TYPE NOMBRE
12     npas EST_DU_TYPE NOMBRE
13     integrale EST_DU_TYPE NOMBRE
14     analytique EST_DU_TYPE NOMBRE
15     pas EST_DU_TYPE NOMBRE
16     x EST_DU_TYPE NOMBRE
17  DEBUT_ALGORITHME
18     //valeur exacte
19     analytique PREND_LA_VALEUR -2.525369
20     //Lecture de l'intervalle
21     LIRE a
22     LIRE b
23     //Nombre de segments
24     LIRE npas
25     //initialisation
26     integrale PREND_LA_VALEUR 0
27     pas PREND_LA_VALEUR (b - a) / npas
28     x PREND_LA_VALEUR a
29     //calcul de l'integrale
30     TANT_QUE (x < b) FAIRE
31         DEBUT_TANT_QUE
32             integrale PREND_LA_VALEUR integrale + pas / 3 * (F1(x) + 4 * F1(x + pas) + F1(x + 2 * pas))
33             x PREND_LA_VALEUR x + 2 * pas
34         FIN_TANT_QUE
35     //résultat
36     AFFICHER "Résultat = "
37     AFFICHER integrale
38     AFFICHER "Résidu = "
39     AFFICHERCALCUL analytique - integrale
40     AFFICHER "Précision = "
41     AFFICHERCALCUL (analytique - integrale) / analytique * 100
42  FIN_ALGORITHME
43
44  // Fonction numérique utilisée :
45  F1(x)=(x * x - 3 * x - 6) * exp(-x)

```



## Algorithme 2.21: Procédé d'orthogonalisation de Gram-Schmidt.

```

1  // schmidt - 13.03.2014
2
3  // *****
4  // Procédé d'orthogonalisation de Gram-Schmidt.
5  // Soit u et v deux vecteurs. On cherche le vecteur vp le plus proche de v orthogonal à u.
6  // *****
7
8  VARIABLES
9      i EST_DU_TYPE NOMBRE
10     u EST_DU_TYPE LISTE
11     v EST_DU_TYPE LISTE
12     vp EST_DU_TYPE LISTE
13     scalaire EST_DU_TYPE NOMBRE
14     normu EST_DU_TYPE NOMBRE
15 DEBUT_ALGORITHME
16     //vecteur u
17     u[1] PREND_LA_VALEUR 1
18     u[2] PREND_LA_VALEUR 0
19     u[3] PREND_LA_VALEUR 0
20     normu PREND_LA_VALEUR sqrt(u[1]*u[1] + u[2]*u[2] + u[3]*u[3])
21     //vecteur v
22     v[1] PREND_LA_VALEUR 1
23     v[2] PREND_LA_VALEUR 2
24     v[3] PREND_LA_VALEUR 3
25     //calcul du produit scalaire
26     scalaire PREND_LA_VALEUR u[1] * v[1] + u[2] * v[2] + u[3] * v[3]
27     AFFICHER "u.v = "
28     AFFICHER scalaire
29     //orthogonalisation de schmidt
30     SI (scalaire != 0) ALORS
31         DEBUT_SI
32             POUR i ALLANT_DE 1 A 3
33                 DEBUT_POUR
34                     vp[i] PREND_LA_VALEUR v[i] - scalaire / normu * u[i]
35                 FIN_POUR
36             //verification
37             AFFICHER "u.vp = "
38             AFFICHERCALCUL u[1] * vp[1] + u[2] * vp[2] + u[3] * vp[3]
39             AFFICHER "vp = "
40             POUR i ALLANT_DE 1 A 3
41                 DEBUT_POUR
42                     AFFICHER vp[i]
43                     AFFICHER " "
44                 FIN_POUR
45             FIN_SI
46 FIN_ALGORITHME

```



---

# Programmes en Fortran

---

## 3.1 Lire – écrire – compter

Code FORTRAN 3.1: Calcul de l'aire d'un rectangle.

```
1  ! aire_rectangle - 11.03.2014
2
3  ! *****
4  ! Calcul de l'aire d'un rectangle
5  ! *****
6
7  program aire_rectangle
8
9      implicit none
10
11     ! declaration
12     real :: largeur, longueur
13
14     ! Lecture des variables
15     write(*,*) "Entrer la largeur"
16     read(*,*) largeur
17     write(*,*) "largeur = ", largeur
18
19     write(*,*) "Entrer la longueur"
20     read(*,*) longueur
21     write(*,*) "longueur = ", longueur
22
23     ! affiche le resultat
24     write(*,*) "Aire du rectangle = ", largeur * longueur
25
26 end program aire_rectangle
```

## Code FORTRAN 3.2: Calcul du reste de la division de deux entiers.

```
1  ! reste - 11.03.2014
2
3  ! *****
4  ! Calcul du reste de la division de deux entiers
5  ! *****
6
7  program prog_reste
8
9      implicit none
10
11     integer :: dividende, diviseur, reste
12
13     ! lecture du dividende
14     write(*,*) "Dividende = "
15     read(*,*) dividende
16     write(*,*) "Dividende = ", dividende
17
18     ! lecture du diviseur
19     write(*,*) "Diviseur = "
20     read(*,*) diviseur
21     write(*,*) "Diviseur = ", diviseur
22
23     ! calcul du reste
24     reste = mod(dividende, diviseur)
25
26     ! affichage du resultat
27     write(*,*) "Reste = ", reste
28
29 end program prog_reste
```

## Code FORTRAN 3.3: Calcul du périmètre d'un cercle.

```
1  ! perimetre - 11.03.2014
2
3  ! *****
4  ! Calcul du périmètre d'un cercle
5  ! *****
6
7  program prog_perimetre
8
9      implicit none
10
11     ! parametres
12     double precision, parameter :: pi = 3.141592653
13
14     ! variables
15     double precision :: rayon, perimetre
16
17     ! lecture du rayon
18     write(*, *) "Rayon = "
19     read(*, *) rayon
20     write(*, *) "Rayon = ", rayon
21
22     ! calcule le perimetre
23     perimetre = 2. * pi * rayon
24     write(*, *) "Perimetre = ", perimetre
25
26 end program prog_perimetre
```

## 3.2 Faire une condition

Code FORTRAN 3.4: Affiche le nombre le plus grand.

```
1  ! le_plus_grand - 11.03.2014
2
3  ! *****
4  ! Lire x et y et dire lequel est le plus grand
5  ! *****
6
7  program le_plus_grand
8
9      implicit none
10
11     real :: x, y
12
13     ! lecture de x et y
14     write(*, *) "Entrer x : "
15     read(*, *) x
16     write(*, *) "x = ", x
17
18     write(*, *) "Entrer y : "
19     read(*, *) y
20     write(*, *) "y = ", y
21
22     ! test entre x et y
23     if (x > y) then
24         write(*, *) "x est plus grand"
25     else if (x < y) then
26         write(*, *) "y est plus grand"
27     else
28         write(*, *) "x et y sont égaux"
29     end if
30 end program le_plus_grand
```

## Code FORTRAN 3.5: Calcul des racines d'un polynôme de degré 2.

```

1  ! discriminant - 11.03.2014
2
3  ! *****
4  ! Calcul des racines réelles d'un polynome de degré 2
5  ! *****
6
7  program discriminant
8
9      implicit none
10
11     ! déclaration
12     real :: a, b, c, delta
13
14     write(*, *) "On va résoudre l'équation  $a \cdot x^2 + b \cdot x + c = 0$ "
15
16     ! Lecture des variables
17     write(*, *) "Entrer a"
18     read(*, *) a
19     write(*, *) "a = ", a
20
21     write(*, *) "Entrer b"
22     read(*, *) b
23     write(*, *) "b = ", b
24
25     write(*, *) "Entrer c"
26     read(*, *) c
27     write(*, *) "c = ", c
28
29     ! Calcul du discriminant
30     delta = b**2 - 4 * a * c
31     write(*, *) "Delta = ", delta
32
33     ! Test du discriminant
34     if (delta > 0.) then
35         write(*, *) "L'équation a deux solutions réelles :"
36         write(*, *) (-b + sqrt(delta)) / (2.0 * a)
37         write(*, *) (-b - sqrt(delta)) / (2.0 * a)
38     else if (delta < 0.) then
39         write(*, *) "L'équation a deux solutions complexes"
40     else
41         write(*, *) "L'équation a une seule solution :"
42         write(*, *) -b / (2.0 * a)
43     end if
44
45 end program discriminant

```

Code FORTRAN 3.6: Calcul de la racine carré d'un nombre.

```

1  ! racine_x - 11.03.2014
2
3  ! *****
4  ! Calcul de la racine carré de x si x est positif.
5  ! *****
6
7  program racine_x
8
9      implicit none
10
11     real :: x
12
13     ! lecture de x
14     write(*, *) "x = "
15     read(*, *) x
16     write(*, *) "x = ", x
17
18     ! test de la valeur de x et calcul de sqrt(x)
19     if (x >= 0) then
20         write(*, *) "racine(x) = ", sqrt(x)
21     else
22         write(*, *) "x est négatif"
23     end if
24
25 end program racine_x

```

### 3.3 Répéter une action

Code FORTRAN 3.7: Illustration de l'utilisation d'une boucle POUR

```

1  ! calc_val_fonction_1 - 11.03.2014
2
3  ! *****
4  ! calcule des valeurs de la fonction  $f(x) = x^2$  pour x dans [-5 ; 5]
5  ! *****
6
7  program calc_val_fonction_1
8
9      implicit none
10
11     ! declaration
12     integer :: x
13
14     ! boucle sur les valeurs de x
15     do x = -5, 5, 1
16         write(*, *) x, x**2
17     end do
18
19 end program calc_val_fonction_1

```



Code FORTRAN 3.8: Illustration de l'utilisation d'une boucle TANT\_QUE.

```
1  ! calc_val_fonction_2 - 11.03.2014
2
3  ! *****
4  ! calcule des valeurs de la fonction  $f(x) = x^2$  pour x dans [-5 ; 5]
5  ! *****
6
7  program calc_val_fonction_2
8
9      implicit none
10
11      ! declaration
12      real :: x, pas
13
14      ! initialisation
15      x = -5.0
16      pas = 0.5
17
18      ! boucle sur les valeurs de x
19      do while (x <= 5.0)
20          write(*, *) x, x**2
21          x = x + pas
22      end do
23
24  end program calc_val_fonction_2
```

## 3.4 Variable indicée

### 3.4.1 Déclaration statique des tableaux

Code FORTRAN 3.9: Création d'une liste contenant les premiers entiers impairs (déclaration statique).

```

1  ! liste_impair - 11.03.2014
2
3  ! *****
4  ! Remplissage et affichage d'une liste des 20 premiers entiers impairs
5  ! tableau statique
6  ! *****
7
8  program liste_impair
9
10     implicit none
11
12     integer                :: i
13     ! nombre de valeurs
14     integer, parameter    :: n = 20
15     ! declaration d'un tableau statique
16     integer, dimension(1:n) :: x
17
18     ! Remplissage de la liste
19     do i = 1, n
20         x(i) = 2 * i - 1
21     end do
22
23     ! affichage de la liste
24     write(*, *) "Liste des ", n, " premiers nombres impaires"
25     do i = 1, n
26         write(*, *) i, x(i)
27     end do
28
29     ! affichage 10 par ligne
30     write(*, "(10i5)") (x(i), i = 1, n)
31
32 end program liste_impair

```

Code FORTRAN 3.10: Création d'une liste de nombres pseudo-aléatoires (déclaration statique).

```
1  ! liste_random - 11.03.2014
2
3  ! *****
4  ! Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
5  ! tableau statique
6  ! *****
7
8  program liste_random
9
10     implicit none
11
12     integer          :: i
13     real             :: x
14     integer, parameter :: n = 15
15     real, dimension(1:n) :: maListe
16
17     ! Remplissage et affichage de la liste
18     do i = 1, n
19         call random_number(x)
20         maListe(i) = 2.0 * x - 1.0
21         write(*, *) i, maListe(i)
22     end do
23
24     ! plus direct
25     call random_number(maListe(:))
26     maListe(:) = 2.0 * maListe(:) - 1.0
27
28 end program liste_random
```

### 3.4.2 Déclaration dynamique des tableaux

Code FORTRAN 3.11: Création d'une liste contenant les premiers entiers impairs (déclaration dynamique).

```

1  ! liste_impair - 11.03.2014
2
3  ! *****
4  ! Remplissage et affichage d'une liste des 20 premiers entiers impairs
5  ! tableau dynamique
6  ! *****
7
8  program liste_impair
9
10     implicit none
11
12     integer :: i, n
13     ! declaration d'un tableau dynamique
14     integer, dimension(:), allocatable :: x
15
16     ! lecture du nombre de valeurs
17     write(*, *) "Nombre de nombres impairs :"
18     read(*, *) n
19     write(*, *) "n = ", n
20
21     ! allocation de la mémoire pour le tableau
22     allocate(x(1:n))
23
24     ! Remplissage de la liste
25     do i = 1, n
26         x(i) = 2 * i - 1
27     end do
28
29     ! affichage 10 par ligne
30     write(*, "(10i5)") (x(i), i = 1, n)
31
32     ! libération de la memoire
33     deallocate(x)
34
35 end program liste_impair

```

Code FORTRAN 3.12: Création d'une liste de nombres pseudo-aléatoires (déclaration dynamique).

```
1  ! liste_random - 11.03.2014
2
3  ! *****
4  ! Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
5  ! tableau dynamique
6  ! *****
7
8  program liste_random
9
10     implicit none
11
12     integer                :: i, n
13     real, dimension(:), allocatable :: maListe
14
15     ! nombre de valeurs
16     write(*, *) "Nombres de valeurs : "
17     read(*, *) n
18     write(*, *) "n = ", n
19
20     ! réservation mémoire
21     allocate(maListe(1:n))
22
23     ! Remplissage de la liste
24     call random_number(maListe(:))
25     maListe(:) = 2.0 * maListe(:) - 1.0
26
27     ! affichage des nombres
28     do i = 1, n
29         write(*, *) i, maListe(i)
30     end do
31
32 end program liste_random
```

### 3.5 Sous programme

Code FORTRAN 3.13: Utilisation de l'écriture d'une fonction et de l'appel à cette fonction.

```

1  ! calc_val_fonction_3 - 11.03.2014
2
3  ! *****
4  ! calcule des valeurs de la fonction  $f(x) = x^2$  pour  $x$  dans  $[-5 ; 5]$ 
5  ! *****
6
7  program calc_val_fonction_3
8
9      implicit none
10
11     ! déclaration
12     real :: x, pas
13
14     ! initialisation
15     x = -5.0
16     pas = 0.5
17
18     ! boucle sur les valeurs de x
19     do while (x <= 5)
20         write(*, *) x, f(x)
21         x = x + pas
22     end do
23
24     contains
25
26     ! fonction numerique
27     real function f(x)
28
29         implicit none
30
31         real :: x
32
33         f = x**2
34
35     end function f
36
37 end program calc_val_fonction_3

```

## 3.6 Premiers petits programmes

Code FORTRAN 3.14: Calcul de la factorielle d'un nombre entier.

```
1  ! factorielle - 11.03.2014
2
3  ! *****
4  ! Calcul de factorielle n
5  ! *****
6
7  program factorielle
8
9      implicit none
10
11     integer :: i, n, facto
12
13     ! lecture de n
14     write(*, *) "Entrer n :"
15     read(*, *) n
16     write(*, *) "n = ", n
17
18     ! initialiation
19     facto = 1
20
21     ! calcul de la factorielle
22     do i = 2, n
23         facto = facto * i
24     end do
25
26     ! affichage resultat
27     write(*, *) "Résultat = ", facto
28
29 end program factorielle
```

## Code FORTRAN 3.15: Calcul d'un produit.

```
1  ! produit - 11.03.2014
2
3  ! *****
4  ! Calcul du produit des n premiers entiers impairs
5  ! *****
6
7  program prog_produit
8
9      implicit none
10
11     integer :: i, n, produit
12
13     ! lecture de n
14     write(*, *) "Entrer n : "
15     read(*,*) n
16     write(*, *) "Calcul du produit des premiers entiers impairs entre 1 et ", n
17
18     ! initialisation
19     produit = 1
20
21     ! boucle
22     do i = 3, n, 1
23         if (mod(i, 2) == 1) then
24             produit = produit * i
25         end if
26     end do
27     write(*, *) "produit = ", produit
28
29     ! autre boucle
30     produit = 1
31     do i = 3, n, 2
32         produit = produit * i
33     end do
34     write(*, *) "produit = ", produit
35
36 end program prog_produit
```



Code FORTRAN 3.16: Calcul d'une somme.

```
1  ! somme - 11.03.2014
2
3  ! *****
4  ! Calcul de la somme des n premiers entiers
5  ! *****
6
7  program prog_somme
8
9      implicit none
10
11     integer :: i, n, somme
12
13     ! lecture de n
14     write(*, *) "n ="
15     read(*, *) n
16     write(*, *) "n = ", n
17
18     ! initialisation
19     somme = 0
20
21     ! boucle
22     do i = 1, n, 1
23         somme = somme + i
24     end do
25
26     ! résultat
27     write(*, *) "somme = ", somme
28
29 end program prog_somme
```

## Code FORTRAN 3.17: Calcul itératif de la décroissance d'une population.

```

1  ! population - 11.03.2014
2
3  ! *****
4  ! Une population est réduite de 40% tous les 3 ans.
5  ! Au bout de combien d'années la population est négligeable
6  ! (inférieure à 0.1% de la population initiale) ?
7  ! *****
8
9  program prog_population
10
11     implicit none
12
13     integer :: an
14     real    :: population, seuil, perte
15
16     ! initialisation
17     population = 100.0
18     an = 0
19     write(*, *) an, population
20
21     ! paramètres
22     perte = 0.4
23     seuil = 0.1 / 100. * population
24
25     ! boucle sur les années
26     do while (population > seuil)
27         an = an + 3
28         population = population * (1.0 - perte)
29         write(*, *) an, population
30     end do
31
32     ! résultats
33     write(*, *) "Au bout de ", an, " ans, la population est inférieure à 0.1% de la population initiale"
34
35 end program prog_population

```

Code FORTRAN 3.18: Recherche du maximum et du minimum dans une liste.

```

1  ! minmax - 11.03.2014
2
3  ! *****
4  ! Recherche du maximum et du minimum dans une liste pseudo-aléatoire
5  ! *****
6
7  program minimax
8
9      implicit none
10
11     integer, parameter :: n = 35
12     integer             :: i
13     real                :: mini, maxi
14     real, dimension(1:n) :: maListe
15
16     ! Remplissage d'une liste pseudo-aléatoire de nombres entre 0 et 100
17     call random_number(maListe(:))
18     maListe(:) = 100.0 * maListe(:)
19     write(*, "(10F8.3)") (maListe(i), i = 1, n)
20
21     ! initialisation
22     mini = maListe(1)
23     maxi = maListe(1)
24
25     ! recherche du maximum et du minimum
26     do i = 1, n
27         ! recherche minimum
28         if (maListe(i) < mini) then
29             mini = maListe(i)
30         end if
31
32         ! recherche maximum
33         if (maListe(i) > maxi) then
34             maxi = maListe(i)
35         end if
36     end do
37
38     ! résultats
39     write(*, *) "Résultats :"
40     write(*, *) "maximum = ", maxi
41     write(*, *) "minimum = ", mini
42
43     ! avec les fonctions internes min(), max()
44     mini = maListe(1)
45     maxi = maListe(1)
46
47     do i = 1, n
48         mini = min(mini, maListe(i))
49         maxi = max(maxi, maListe(i))
50     end do
51
52     write(*, *) "maximum = ", maxi
53     write(*, *) "minimum = ", mini
54
55     ! avec les fonctions internes maxval(), minval()
56     mini = minval(maListe(:))
57     maxi = maxval(maListe(:))
58

```

## Code FORTRAN 3.19: Marche aléatoire d'un point dans un plan 2D.

```

1  ! marche_aleatoire - 11.03.2014
2
3  ! *****
4  ! Marche aléatoire dans une plan 2D
5  ! *****
6
7  program marche_aleatoire
8
9      implicit none
10
11     integer, parameter          :: d = 2      ! dimension
12     integer                     :: i, k, npas
13     real                        :: amplitude
14     real, dimension(d)          :: w
15     real, dimension(:, :), allocatable :: x
16
17     ! paramètres de la marche
18     write(*, *) "Nombre de pas : "
19     read(*, *) npas
20     write(*, *) "npas = ", npas
21
22     write(*, *) "Amplitude des déplacement : "
23     read(*, *) amplitude
24     write(*, *) "amplitude = ", amplitude
25
26     ! allocation memoire
27     allocate(x(npas, d))
28
29     ! initialisation
30     do k = 1, d
31         x(1, k) = 0.0
32     end do
33
34     ! marche aleatoire
35     do i = 2, npas
36         do k = 1, d
37             call random_number(w(k))
38             w(k) = 2.0 * w(k) - 1.0
39             x(i, k) = x(i - 1, k) + amplitude * w(k)
40         end do
41     end do
42
43     ! enregistrement de la trajectoire dans un fichier
44     open(unit = 10, file = "traj.dat", action = "write")
45     do i = 1, npas
46         write(10, *) (x(i, k), k = 1, d)
47     end do
48     close(unit = 10)
49
50 end program marche_aleatoire

```

Code FORTRAN 3.20: Calcul du nombre  $\pi$  par la méthode Monte Carlo.

```
1  program prog_pi
2
3  implicit none
4
5  integer                :: i, n, ntirage
6  double precision       :: x, y
7  double precision, parameter :: pi = 3.141592653589793d0
8
9  ! lecture du nombre de tirage
10 write(*, *) "Nombre de tirage : "
11 read(*, *) ntirage
12 write(*, *) "ntirage = ", ntirage
13
14 ! initialisation
15 n = 0
16
17 ! calcul du nombre pi
18 do i = 1, ntirage, 1
19     call random_number(x)
20     call random_number(y)
21
22     if (x**2 + y**2 < 1.d0) then
23         n = n + 1
24     end if
25 end do
26
27 write(*, *) "pi = ", 4.0 * dble(n) / dble(ntirage)
28 write(*, *) "% d'erreur = ", (pi - 4.d0 * dble(n) / dble(ntirage)) / pi * 100.d0
29
30 end program prog_pi
```

## Code FORTRAN 3.21: Intégration par la méthode des trapèzes.

```

1  ! trapeze - 11.03.2014
2
3  ! *****
4  ! Intégration par la méthode des trapèzes. L'intégration analytique
5  ! de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
6  ! application numérique -2.525369
7  ! *****
8
9  program trapeze
10
11     implicit none
12
13     integer      :: i, npas
14     real         :: a, b, pas, integrale, x
15
16     ! valeur exacte
17     real, parameter :: analytique = -2.525369
18
19     ! Lecture de l'intervalle
20     write(*, *) "a = "
21     read(*, *) a
22     write(*, *) "a = ", a
23
24     write(*, *) "b = "
25     read(*, *) b
26     write(*, *) "b = ", b
27
28     ! Nombre de segments
29     write(*, *) "npas = "
30     read(*, *) npas
31     write(*, *) "npas = ", npas
32
33     ! initialisation
34     integrale = 0.0
35     pas = (b - a) / real(npas)
36
37     ! calcul de l'integrale
38     do i = 1, npas
39         x = a + (i - 1) * pas
40         integrale = integrale + pas * (f(x) + f(x + pas)) / 2.0
41     end do
42
43     ! résultats
44     write(*, *) "Résultat = ", integrale
45     write(*, *) "Résidu = ", analytique - integrale
46     write(*, *) "Précision = ", (analytique - integrale) / analytique * 100.0
47
48     contains
49
50     ! fonction numérique utilisée
51     real function f(x)
52         implicit none
53         real :: x
54         f = (x**2 - 3 * x - 6) * exp(-x)
55     end function f
56
57 end program trapeze

```

## Code FORTRAN 3.22: Intégration par la méthode de Simpson.

```

1  ! simpson - 11.03.2014
2
3  ! *****
4  ! Intégration par la méthode de Simpson. L'intégration analytique
5  ! de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
6  ! application numérique -2.525369
7  ! *****
8
9  program simpson
10
11     implicit none
12
13     integer      :: npas
14     real         :: a, b, pas, integrale, x
15
16     ! valeur exacte
17     real, parameter :: analytique = -2.525369
18
19     ! Lecture de l'intervalle
20     write(*, *) "a = "
21     read(*, *) a
22     write(*, *) "a = ", a
23
24     write(*, *) "b = "
25     read(*, *) b
26     write(*, *) "b = ", b
27
28     ! Nombre de segments
29     write(*, *) "npas = "
30     read(*, *) npas
31     write(*, *) "npas = ", npas
32
33     ! initialisation
34     integrale = 0.0
35     pas = (b - a) / real(npas)
36     x = a
37
38     ! calcul de l'integrale
39     do while (x < b)
40         integrale = integrale + pas / 3.0 * (f(x) + 4.0 * f(x + pas) + f(x + 2.0 * pas))
41         x = x + 2.0 * pas
42     end do
43
44     ! résultats
45     write(*, *) "Résultat = ", integrale
46     write(*, *) "Résidu = ", analytique - integrale
47     write(*, *) "Précision = ", (analytique - integrale) / analytique * 100.0
48
49     contains
50
51     ! fonction numérique utilisée
52     real function f(x)
53         implicit none
54         real :: x
55         f = (x**2 - 3 * x - 6) * exp(-x)
56     end function f
57

```

## Code FORTRAN 3.23: Procédé d'orthogonalisation de Gramm-Schmidt.

```

1  ! schmidt - 13.03.2014
2
3  ! *****
4  ! Procédé d'orthogonalisation de Gram-Schmidt.
5  ! Soit u et v deux vecteurs. On cherche le vecteur vp
6  ! le plus proche de v orthogonal à u.
7  ! *****
8
9  program schmidt
10
11     implicit none
12
13     integer          :: i
14     real             :: scalaire, normu
15     real, dimension(3) :: u, v, vp
16
17     ! vecteur u
18     u(1) = 1.0
19     u(2) = 0.0
20     u(3) = 0.0
21     normu = sqrt(sum((u(:))**2))
22
23     ! vecteur v
24     v(1) = 1.0
25     v(2) = 2.0
26     v(3) = 3.0
27
28     ! calcul du produit scalaire
29     scalaire = dot_product(u, v)
30     write(*, *) "u.v = ", scalaire
31
32     ! orthogonalisation de schmidt
33     if (scalaire /= 0) then
34         vp(:) = v(:) - scalaire / normu * u(:)
35         ! verification
36         write(*, *) "u.vp = ", dot_product(u, vp)
37         write(*, *) "vp = ", (vp(i), i = 1, 3)
38     end if
39
40 end program schmidt

```



# Programmes en Python

---

Code Python 4.1: Structure générale adoptée pour l'écriture des programmes en Python

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ Présentation du code """
5
6  def fonction(argument1, argument2):
7      """ description de la fonction """
8
9      # programmation de la fonction
10
11     return "une valeur"
12
13 if __name__ == "__main__":
14     # instructions à exécuter, par exemple
15     #     * lecture des parametres
16     #     * appel de la fonction
17     fonction()
```

## 4.1 Lire – écrire – compter

### 4.1.1 Versions sans fonction

Code Python 4.2: Calcul de l'aire d'un rectangle.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ aire_rectangle - 11.03.2014
5
6  Calcul de l'aire d'un rectangle """
7
8  # lecture des variables
9  largeur = float(raw_input("largeur = "))
10 print("largeur = {0}".format(largeur))
11
12 longueur = float(raw_input("longueur = "))
13 print("longueur = {0}".format(longueur))
14
15 # affiche le résultats
16 print("Aire du rectangle = {0} ".format(largeur * longueur))
```

Code Python 4.3: Calcul du reste de la division de deux entiers.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ reste - 11.03.2014
5
6  Calcul du reste de la division de deux entiers """
7
8  # lecture des valeurs
9  dividende = int(raw_input("entrer le dividende : "))
10 print("dividende = {0}".format(dividende))
11
12 diviseur = int(raw_input("entre le diviseur : "))
13 print("diviseur = {0}".format(diviseur))
14
15 # calcul du reste
16 reste = dividende % diviseur
17
18 # affichage du résultat
19 print("reste = {0}".format(reste))
```

Code Python 4.4: Calcul du périmètre d'un cercle.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ perimetre - 11.03.2014
5
6  Calcul du périmètre d'un cercle """
7
8  import math
9
10 # lecture du rayon
11 rayon = float(raw_input("entrer le rayon : "))
12 print ("rayon = {0}".format(rayon))
13
14 # calcul du perimetre
15 perimetre = 2.0 * math.pi * rayon
16
17 # affichage du résultat
18 print ("périmètre = {0}".format(perimetre))

```

## 4.1.2 Versions avec une fonction

Code Python 4.5: Calcul de l'aire d'un rectangle.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ aire_rectangle - 11.03.2014 """
5
6  def aire_rectangle(largeur, longueur):
7      """ Calcul de l'aire d'un rectangle """
8      return largeur * longueur
9
10 if __name__ == "__main__":
11     # lecture des variables
12     largeur = float(raw_input("largeur = "))
13     print ("largeur = {0}".format(largeur))
14
15     longueur = float(raw_input("largeur = "))
16     print ("longueur = {0}".format(longueur))
17
18     # arriche le résultats
19     print ("Aire du rectangle = {0} ".format(aire_rectangle(largeur, longueur)))

```

Code Python 4.6: Calcul du reste de la division de deux entiers.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ reste - 11.03.2014 """
5
6  def prog_reste(dividende, diviseur):
7      """ Calcul du reste de la division de deux entiers """
8
9      reste = dividende % diviseur
10
11     print("reste = {0}".format(reste))
12
13 if __name__ == "__main__":
14     # lecture des valeurs
15     dividende = int(raw_input("entrer le dividende : "))
16     print("dividende = {0}".format(dividende))
17
18     diviseur = int(raw_input("entre le diviseur : "))
19     print("diviseur = {0}".format(diviseur))
20
21     prog_reste(dividende, diviseur)

```

Code Python 4.7: Calcul du périmètre d'un cercle.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ perimetre - 11.03.2014 """
5
6  import math
7
8  def perimetre(rayon):
9      """ Calcul du périmètre d'un cercle """
10     return 2.0 * math.pi * rayon
11
12 if __name__ == "__main__":
13     # lecture du rayon
14     rayon = float(raw_input("entrer le rayon : "))
15     print("rayon = {0}".format(rayon))
16
17     # affichage du résultat
18     print("périmètre = {0}".format(perimetre(rayon)))

```

## 4.2 Faire une condition

### 4.2.1 Versions sans fonction

Code Python 4.8: Affiche le nombre le plus grand.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ le_plus_grand - 11.03.2014
5
6  Lire x et y et dire lequel est le plus grand """
7
8  # lecture de x et y
9  x = float(raw_input("entrer x : "))
10 print("x = {0}".format(x))
11
12 y = float(raw_input("entrer y : "))
13 print("y = {0}".format(y))
14
15 # test entre x et y
16 if x > y:
17     print("x est plus grand")
18     print("le plus grand = {0}".format(x))
19 elif y > x:
20     print("y est plus grand")
21     print("le plus grand = {0}".format(y))
22 else:
23     print("x et y sont égaux")
24     print("x = {0}\t y = {1}".format(x, y))
```

## Code Python 4.9: Calcul des racines d'un polynôme de degré 2.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ discriminant - 11.03.2014
5
6  Calcul des racines réelles d'un polynome de degré 2 """
7
8  from math import sqrt
9
10 print("On va résoudre l'équation  $a \cdot x^2 + b \cdot x + c = 0$ ")
11
12 # lecture des variables
13 a = float(raw_input("entrer a : "))
14 print("a = {0}".format(a))
15
16 b = float(raw_input("entrer b : "))
17 print("b = {0}".format(b))
18
19 c = float(raw_input("entrer c : "))
20 print("c = {0}".format(c))
21
22 # Calcul du discriminant
23 delta = b**2 - 4. * a * c
24 print("delta = {0}".format(delta))
25
26 # Test du discriminant
27 if delta > 0:
28     print("L'équation a deux solutions")
29     print("x1 = {0}".format((-b - sqrt(delta)) / (2. * a)))
30     print("x2 = {0}".format((-b + sqrt(delta)) / (2. * a)))
31 elif delta < 0.:
32     print("L'équation a deux solutions complexes")
33 else:
34     print("L'équation a une seule solution")
35     print("x = {0}".format(-b / (2.0 * a)))
```

Code Python 4.10: Calcul de la racine carré d'un nombre.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ racine_x - 11.03.2014
5
6  Calcul de la racine carré de x si x est positif. """
7
8  from math import sqrt
9
10 # lecture de x
11 x = float(raw_input("entrer x : "))
12 print("x = {0}".format(x))
13
14 # test de la valeur de x et calcul de sqrt(x)
15 if x >= 0:
16     print("RACINE(x) = {0}".format(sqrt(x)))
17 else:
18     print("x est négatif")
```

### 4.2.2 Versions avec une fonction

Code Python 4.11: Affiche le nombre le plus grand.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ le_plus_grand - 11.03.2014 """
5
6  def le_plus_grand(x, y):
7      """ donne le nombre le plus grand """
8
9      # test entre x et y
10     if x > y:
11         print("x est plus grand")
12         print("le plus grand = {0}".format(x))
13     elif y > x:
14         print("y est plus grand")
15         print("le plus grand = {0}".format(y))
16     else:
17         print("x et y sont égaux")
18         print("x = {0}\t y = {1}".format(x, y))
19
20 if __name__ == "__main__":
21     # lecture de x et y
22     x = float(raw_input("entrer x : "))
23     print("x = {0}".format(x))
24
25     y = float(raw_input("entrer y : "))
26     print("y = {0}".format(y))
27
28     # appel de la fonction
29     le_plus_grand(x, y)
```



Code Python 4.12: Calcul des racines d'un polynôme de degré 2.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ discriminant - 11.03.2014 """
5
6  from math import sqrt
7
8  def discriminant(a, b, c):
9      """ Calcul des racines réelles d'un polynome de degré 2 """
10
11     # Calcul du discriminant
12     delta = b**2 - 4. * a * c
13     print("delta = {0}".format(delta))
14
15     # Test du discriminant
16     if delta > 0:
17         print("L'équation a deux solutions")
18         print("x1 = {0}".format((-b - sqrt(delta)) / (2. * a)))
19         print("x2 = {0}".format((-b + sqrt(delta)) / (2. * a)))
20     elif delta < 0.:
21         print("L'équation a deux solutions complexes")
22     else:
23         print("L'équation a une seule solution")
24         print("x = {0}".format(-b / (2.0 * a)))
25
26  if __name__ == "__main__":
27     print("On va résoudre l'équation a*x^2 + b*x + c = 0")
28
29     # lecture des variables
30     a = float(raw_input("entrer a : "))
31     print("a = {0}".format(a))
32
33     b = float(raw_input("entrer b : "))
34     print("b = {0}".format(b))
35
36     c = float(raw_input("entrer c : "))
37     print("c = {0}".format(c))
38
39     discriminant(a, b, c)
```

Code Python 4.13: Calcul de la racine carré d'un nombre.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ racine_x - 11.03.2014 """
5
6  from math import sqrt
7
8  def racine(x):
9      """ Calcul de la racine carré de x si x est positif. """
10
11     if x >= 0:
12         print("RACINE(x) = {0}".format(sqrt(x)))
13     else:
14         print("x est négatif")
15
16 if __name__ == "__main__":
17     # lecture de x
18     x = float(raw_input("entrer x : "))
19     print("x = {0}".format(x))
20
21     racine(x)

```

### 4.3 Répéter une action

Code Python 4.14: Illustration de l'utilisation d'une boucle POUR

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ calc_val_fonction_1 - 11.03.2014 """
5
6  def calc_val_fonction_1():
7      """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """
8
9      for x in range(-5, 6, 1):
10         print(x**2)
11
12 if __name__ == "__main__":
13     calc_val_fonction_1()

```

Code Python 4.15: Illustration de l'utilisation d'une boucle TANT\_QUE.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ calc_val_fonction_2 - 11.03.2014 """
5
6  def calc_val_fonction_2():
7      """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """
8
9      # initialisation
10     x = -5.0
11     pas = 0.5
12
13     # boucle sur les valeurs de x
14     while x <= 5:
15         print(x**2)
16         x += pas
17
18 if __name__ == "__main__":
19     calc_val_fonction_2()
```

## 4.4 Variable indicée

Code Python 4.16: Création d'une liste contenant les premiers entiers impairs.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ liste_impair - 11.03.2014 """
5
6  def liste_impair(n = 20):
7      """ Remplissage et affichage d'une liste des n premiers
8          entiers impairs """
9
10     # creation d'une liste
11     x = list()
12
13     # remplissage de la liste
14     for i in range(n):
15         x.append(2 * i + 1)
16
17     # ou
18     x = [2 * i + 1 for i in range(n)]
19
20     # affichage de la liste
21     print("liste des {0} premiers nombres impairs".format(n))
22     for i in range(n):
23         print(x[i])
24
25 if __name__ == "__main__":
26     # lecture du nombre d'entiers
27     n = int(raw_input("entrer n : "))
28     print("n = {0}".format(n))
29
30     liste_impair(n)
```

Code Python 4.17: Création d'une liste de nombres pseudo-aléatoires.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ liste_random - 11.03.2014 """
5
6  from random import random
7
8  def liste_random(n = 20):
9      """ Remplissage et affichage d'une liste aléatoire
10         de nombre entre -1 et 1 """
11
12         # creation de la liste
13         maListe = list()
14
15         # remplissage et affichage de la liste
16         for i in range(n):
17             x = 2. * random() - 1.
18             maListe.append(x)
19             print(x)
20
21         # ou
22         maListe = [2. * random() - 1. for i in range(n)]
23
24  if __name__ == "__main__":
25      # lecture du nombre de tirage aleatoire
26      n = int(raw_input("entrer n : "))
27      print("n = {0}".format(n))
28
29      liste_random(n)
```

## 4.5 Sous programme

Code Python 4.18: Utilisation de l'écriture d'une fonction et de l'appel à cette fonction.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ calc_val_fonction_3 - 11.03.2014 """
5
6  def calc_val_fonction_3(pas = 0.1):
7      """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """
8
9      # initialisation
10     x = -5.0
11
12     # boucle sur les valeurs de x
13     while x <= 5:
14         print(f(x))
15         x += pas
16
17 def f(x):
18     """ fonction numérique utilisée """
19     return x**2
20
21 if __name__ == "__main__":
22     pas = 0.5
23     calc_val_fonction_3(pas)
24     #calc_val_fonction_3()
```

## 4.6 Premiers petits programmes

Code Python 4.19: Calcul de la factorielle d'un nombre entier.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ factorielle - 11.03.2014 """
5
6  def factorielle(n):
7      """ Calcul de factorielle n """
8
9      if n == 0:
10         factorielle = 1
11     else:
12         # initialisation
13         factorielle = 1
14
15         # calcul
16         for i in range(2, n + 1):
17             factorielle *= i
18
19     return factorielle
20
21 if __name__ == "__main__":
22     # lecture de n
23     n = int(raw_input("entrer n : "))
24     print("n = {0}".format(n))
25
26     # affichage du résultat
27     print("{0}! = {1}".format(n, factorielle(n)))
```

Code Python 4.20: Calcul d'un produit.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ produit - 11.03.2014 """
5
6  def prog_produit():
7      """ Calcul du produit des n premiers entiers impairs """
8
9      # lecture de n
10     n = int(raw_input("entrer n : "))
11     print("Calcul du produit des entiers impairs entre 1 et {0}".format(n))
12
13     # initialisation
14     produit = 1
15
16     # calcul
17     for i in range(3, n + 1):
18         if i % 2 == 1:
19             produit *= i
20
21     # autre methode
22     produit = 1
23     for i in range(3, n + 1, 2):
24         produit *= i
25
26     # affichage du resultat
27     print("Résultat = {0}".format(produit))
28
29 if __name__ == "__main__":
30     prog_produit()
```



Code Python 4.21: Calcul d'une somme.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ somme - 11.03.2014 """
5
6  def prog_somme(n):
7      """ Calcul de la somme des n premiers entiers """
8
9      # initialisation
10     somme = 0.0
11
12     for i in range(n + 1):
13         somme += i
14
15     # ou somme = sum(range(n + 1))
16
17     return somme
18
19 if __name__ == "__main__":
20     # lecture de n
21     n = int(raw_input("entrer n : "))
22     print("n = {0}".format(n))
23
24     print("Résultat = {0}".format(prog_somme(n)))
```

Code Python 4.22: Calcul itératif de la décroissance d'une population.

```
1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ population - 11.03.2014 """
5
6  def prog_population():
7      """ Une population est réduite de 40% tous les 3 ans.
8          Au bout de combien d'années la population est négligeable
9          (inférieure à 0.1% de la population initiale) ? """
10
11     # initialisation
12     population = 100.0
13     an = 0
14     perte = .4
15     seuil = .1 / 100. * population
16
17     # boucle sur les années
18     while population > seuil:
19         an += 3
20         population *= (1. - perte)
21         print("{0} {1}".format(an, population))
22
23     print("Au bout de {0} ans, la population est inférieure à 0.1% de la population initiale.".format(an))
24
25 if __name__ == "__main__":
26     prog_population()
```

Code Python 4.23: Recherche du maximum et du minimum dans une liste.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ minmax - 11.03.2014 """
5
6  from random import random
7
8  def minimax(maListe):
9      """ Recherche du maximum et du minimum dans une liste pseudo-aléatoire """
10
11     # recherche du maximum et du minimum
12     # initialisation
13     mini = maListe[0]
14     maxi = maListe[0]
15
16     for i in range(n):
17         # recherche du minimum
18         if maListe[i] < mini:
19             mini = maListe[i]
20
21         # recherche du maximum
22         if maListe[i] > maxi:
23             maxi = maListe[i]
24
25     print("maximum = {0}".format(maxi))
26     print("minimum = {0}".format(mini))
27
28     # avec les fonctions min() et max()
29     print("maximum = {0}".format(max(maListe)))
30     print("minimum = {0}".format(min(maListe)))
31
32 if __name__ == "__main__":
33     # nombre de points
34     n = int(raw_input("entrer n : "))
35     print("n = {0}".format(n))
36
37     # remplissage d'une liste pseudo aleatoire de nombres entre 0 et 100
38     maListe = [100. * random() for i in range(n)]
39     # ou avec des entiers
40     # maListe = [randint(0, 100) for i in range(n)]
41
42     minimax(maListe)

```

Code Python 4.24: Marche aléatoire d'un point dans un plan 2D.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ marche_aleatoire - 11.03.2014 """
5
6  from random import random
7
8  def marche_aleatoire():
9      """ Marche aléatoire dans un plan 2D """
10
11     # nombres de pas
12     npas = 1000
13     amplitude = .1
14
15     # liste pour les coordonnées
16     x = list()
17     y = list()
18
19     # initialisation
20     x.append(0.)
21     y.append(0.)
22
23     # marche aléatoire
24     for i in range(npas):
25         wx = 2. * random() - 1.
26         wy = 2. * random() - 1.
27         x.append(x[i] + amplitude * wx)
28         y.append(y[i] + amplitude * wy)
29
30     # représentation avec matplotlib
31     import matplotlib.pyplot as plt
32
33     plt.plot(x, y, "r-", label = "trajectoire")
34     plt.title("Marche aleatoire dans un plan 2D")
35     plt.axes().set_aspect("equal")
36     plt.grid()
37     plt.xlabel("x")
38     plt.ylabel("y")
39     plt.legend()
40     plt.show()
41
42 if __name__ == "__main__":
43     marche_aleatoire()

```

Code Python 4.25: Calcul du nombre  $\pi$  par la méthode Monte Carlo.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ pi - 14.03.2014
5
6      Calcul du nombre pi par Monte Carlo.
7      Principe : La probabilité qu'un point de coordonnées (x,y)
8      avec x et y dans l'intervalle [0;1] soit dans le quart de
9      cercle de centre (0,0) et de rayon 1 est égal au rapport des
10     aires du quart de cercle de rayon 1 et du carré de largeur 1
11     soit pi/4. """
12
13  import math
14  from random import random
15
16  def prog_pi(ntirage = 100):
17      """ calcul du nombre pi """
18
19      # initialisation
20      n = 0
21
22      # calcul monte carlo
23      for i in range(ntirage):
24          x = random()
25          y = random()
26
27          if x**2 + y**2 < 1.0:
28              n += 1
29
30      # affichage des résultats
31      piCalcule = 4.0 * float(n) / float(ntirage)
32      print("pi          = {}".format(piCalcule))
33      print("% d'erreur = {}".format((math.pi - piCalcule) / math.pi * 100))
34
35  if __name__ == "__main__":
36      # lecture du nombre de tirage
37      ntirage = int(raw_input("entrer le nombre de tirage : "))
38      print("ntirage = {}".format(ntirage))
39
40      prog_pi(ntirage)

```

Code Python 4.26: Intégration par la méthode des trapèzes.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ trapeze - 11.03.2014
5
6  Intégration par la méthode des trapèzes. L'intégration analytique
7  de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
8  application numérique -2.525369
9
10 voir également : scipy.integrate.trapz
11 """
12
13 from math import exp
14
15 def trapeze(a, b, npas):
16     """ integration par la methode des trapeze """
17
18     # initialisation
19     integrale = 0.
20     pas = (b - a) / float(npas)
21
22     # calcul de l'integrale
23     for i in range(npas):
24         x = a + float(i) * pas
25         integrale += pas * (f(x) + f(x + pas)) / 2.
26
27     return integrale
28
29 def f(x):
30     """ fonction numerique utilisée """
31     return (x**2 - 3. * x - 6.) * exp(-x)
32
33 if __name__ == "__main__":
34     # lecture de l'intervalle
35     a = float(raw_input("entrer a : "))
36     print("a = {0}".format(a))
37
38     b = float(raw_input("entrer b : "))
39     print("b = {0}".format(b))
40
41     # nombre de segments
42     npas = int(raw_input("entrer le nombre de pas : "))
43     print("npas = {0}".format(npas))
44
45     integrale = trapeze(a, b, npas)
46
47     analytique = -2.525369
48     print("Résultat = {0}".format(integrale))
49     print("Résidu = {0}".format(analytique - integrale))
50     print("Précision = {0}".format((analytique - integrale) / analytique * 100))

```

## Code Python 4.27: Intégration par la méthode de Simpson.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ simpson - 11.03.2014
5
6  Intégration par la méthode de Simpson. L'intégration analytique
7  de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
8  application numérique -2.525369
9
10 voir également : scipy.integrate.simps
11 """
12
13 from math import exp
14
15 def simpson(a, b, npas = 20):
16     """ integration de simpson """
17
18     # initialisation
19     integrale = 0.0
20     pas = (b - a) / float(npas)
21     x = a
22
23     # calcul de l'integrale
24     while x < b:
25         integrale += pas / 3. * (f(x) + 4. * f(x + pas) + f(x + 2. * pas))
26         x += 2. * pas
27
28     return integrale
29
30 def f(x):
31     """ fonction numérique utilisée """
32     return (x**2 - 3 * x - 6) * exp(-x)
33
34 if __name__ == "__main__":
35     # lecture de l'intervalle
36     a = float(raw_input("entrer a : "))
37     print("a = {0}".format(a))
38
39     b = float(raw_input("entrer b : "))
40     print("b = {0}".format(b))
41
42     # nombre de pas
43     npas = int(raw_input("entrer le nombre de pas : "))
44     print("npas = {0}".format(npas))
45
46     integrale = simpson(a, b, npas)
47
48     analytique = -2.525369
49     print("Résultat = {0}".format(integrale))
50     print("Résidu = {0}".format(analytique - integrale))
51     print("Précision = {0}".format((analytique - integrale) / analytique * 100))

```

## Code Python 4.28: Procédé d'orthogonalisation de Gramm-Schmidt.

```

1  #!/usr/bin/env python
2  # -*- coding=utf-8 -*-
3
4  """ schmidt - 13.03.2014 """
5
6  from math import sqrt
7
8  def schmidt() :
9      """ Procédé d'orthogonalisation de Gram-Schmidt.
10         Soit u et v deux vecteurs. On cherche le vecteur vp le plus
11         proche de v orthogonal à u. """
12
13         # vecteur u
14         u = [1., 0., 0.]
15         normu = sqrt(sum([ui**2 for ui in u]))
16
17         # vecteur v
18         v = [1., 2., 3.]
19
20         # calcul du produit scalaire
21         scalaire = sum([ui * vi for ui, vi in zip(u, v)])
22         print("u.v = {0}".format(scalaire))
23
24         # orthogonalisation de schmidt
25         if scalaire != 0.0:
26             vp = [vi - scalaire / normu * ui for ui, vi in zip(u, v)]
27
28             # verification
29             print("u.vp = {0}".format(sum([ui * vpi for ui, vpi in zip(u, vp)])))
30             print("vp = {0}".format(vp))
31
32 if __name__ == "__main__":
33     schmidt()

```



---

# Algorithmes AlgoBox

---

2.1	Calcul de l'aire d'un rectangle. . . . .	9
2.2	Calcul du reste de la division de deux entiers. . . . .	10
2.3	Calcul du périmètre d'un cercle. . . . .	10
2.4	Affiche le nombre le plus grand. . . . .	11
2.5	Calcul des racines d'un polynôme de degré 2. . . . .	12
2.6	Calcul de la racine carré d'un nombre. . . . .	13
2.7	Illustration de l'utilisation d'une boucle POUR . . . . .	13
2.8	Illustration de l'utilisation d'une boucle TANT_QUE. . . . .	14
2.9	Création d'une liste contenant les premiers entiers impairs. . . . .	14
2.10	Création d'une liste de nombres pseudo-aléatoires. . . . .	15
2.11	Utilisation de l'écriture d'une fonction et de l'appel à cette fonction. . . . .	15
2.12	Calcul de la factorielle d'un nombre entier. . . . .	16
2.13	Calcul d'un produit. . . . .	17
2.14	Calcul d'une somme. . . . .	18
2.15	Calcul itératif de la décroissance d'une population. . . . .	19
2.16	Recherche du maximum et du minimum dans une liste. . . . .	20
2.17	Marche aléatoire d'un point dans un plan 2D. . . . .	21
2.18	Calcul du nombre $\pi$ par la méthode Monte Carlo. . . . .	22
2.19	Intégration par la méthode des trapèzes. . . . .	23
2.20	Intégration par la méthode de Simpson. . . . .	24
2.21	Procédé d'orthogonalisation de Gramm-Schmidt. . . . .	25



---

# Codes en FORTRAN

---

3.1	Calcul de l'aire d'un rectangle. . . . .	27
3.2	Calcul du reste de la division de deux entiers. . . . .	28
3.3	Calcul du périmètre d'un cercle. . . . .	29
3.4	Affiche le nombre le plus grand. . . . .	30
3.5	Calcul des racines d'un polynôme de degré 2. . . . .	31
3.6	Calcul de la racine carré d'un nombre. . . . .	32
3.7	Illustration de l'utilisation d'une boucle POUR . . . . .	32
3.8	Illustration de l'utilisation d'une boucle TANT_QUE. . . . .	33
3.9	Création d'une liste contenant les premiers entiers impairs (déclaration statique). . . .	34
3.10	Création d'une liste de nombres pseudo-aléatoires (déclaration statique). . . . .	35
3.11	Création d'une liste contenant les premiers entiers impairs (déclaration dynamique). .	36
3.12	Création d'une liste de nombres pseudo-aléatoires (déclaration dynamique). . . . .	37
3.13	Utilisation de l'écriture d'une fonction et de l'appel à cette fonction. . . . .	38
3.14	Calcul de la factorielle d'un nombre entier. . . . .	39
3.15	Calcul d'un produit. . . . .	40
3.16	Calcul d'une somme. . . . .	41
3.17	Calcul itératif de la décroissance d'une population. . . . .	42
3.18	Recherche du maximum et du minimum dans une liste. . . . .	43
3.19	Marche aléatoire d'un point dans un plan 2D. . . . .	44
3.20	Calcul du nombre $\pi$ par la méthode Monte Carlo. . . . .	45
3.21	Intégration par la méthode des trapèzes. . . . .	46
3.22	Intégration par la méthode de Simpson. . . . .	47
3.23	Procédé d'orthogonalisation de Gramm-Schmidt. . . . .	48



---

# Codes en Python

---

4.1	Structure générale adoptée pour l'écriture des programmes en Python . . . . .	49
4.2	Calcul de l'aire d'un rectangle. . . . .	50
4.3	Calcul du reste de la division de deux entiers. . . . .	50
4.4	Calcul du périmètre d'un cercle. . . . .	51
4.5	Calcul de l'aire d'un rectangle. . . . .	51
4.6	Calcul du reste de la division de deux entiers. . . . .	52
4.7	Calcul du périmètre d'un cercle. . . . .	52
4.8	Affiche le nombre le plus grand. . . . .	53
4.9	Calcul des racines d'un polynôme de degré 2. . . . .	54
4.10	Calcul de la racine carré d'un nombre. . . . .	55
4.11	Affiche le nombre le plus grand. . . . .	56
4.12	Calcul des racines d'un polynôme de degré 2. . . . .	57
4.13	Calcul de la racine carré d'un nombre. . . . .	58
4.14	Illustration de l'utilisation d'une boucle POUR . . . . .	58
4.15	Illustration de l'utilisation d'une boucle TANT_QUE. . . . .	59
4.16	Création d'une liste contenant les premiers entiers impairs. . . . .	60
4.17	Création d'une liste de nombres pseudo-aléatoires. . . . .	61
4.18	Utilisation de l'écriture d'une fonction et de l'appel à cette fonction. . . . .	62
4.19	Calcul de la factorielle d'un nombre entier. . . . .	63
4.20	Calcul d'un produit. . . . .	64
4.21	Calcul d'une somme. . . . .	65
4.22	Calcul itératif de la décroissance d'une population. . . . .	66
4.23	Recherche du maximum et du minimum dans une liste. . . . .	67
4.24	Marche aléatoire d'un point dans un plan 2D. . . . .	68
4.25	Calcul du nombre $\pi$ par la méthode Monte Carlo. . . . .	69
4.26	Intégration par la méthode des trapèzes. . . . .	70
4.27	Intégration par la méthode de Simpson. . . . .	71
4.28	Procédé d'orthogonalisation de Gramm-Schmidt. . . . .	72