18 mars 2014

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

Programming from examples

Chapitre 1

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.

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

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

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

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

    $$\vec{v}' = \vec{v} - \frac{(\vec{u}.\vec{v})}{||\vec{u}||}\vec{u}$$

Programming from examples

Chapitre 2

Algorithme – AlgoBox

2.1 Lire – écrire – compter

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

```
// aire_rectangle  -  11.03.2014

// ****************************************
// Calcul de l'aire d'un rectangle
// ****************************************

VARIABLES
  largeur EST_DU_TYPE NOMBRE
  longueur EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  // Lecture des variables
  LIRE largeur
  LIRE longueur
  //Affiche le résultat
  AFFICHER "Aire du rectangle : "
  AFFICHERCALCUL largeur * longueur
FIN_ALGORITHME
```

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

```
// reste  -  11.03.2014

// ****************************************
// Calcul du reste de la division de deux entiers
// ****************************************

VARIABLES
  dividende EST_DU_TYPE NOMBRE
  diviseur EST_DU_TYPE NOMBRE
  reste EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Lecture des valeurs
  LIRE dividende
  AFFICHER "dividende = "
  AFFICHER dividende
  LIRE diviseur
  AFFICHER "diviseur = "
  AFFICHER diviseur
  //Calcul du reste
  reste PREND_LA_VALEUR dividende % diviseur
  //Affichage du resultat
  AFFICHER "reste = "
  AFFICHER reste
FIN_ALGORITHME
```

Algorithme 2.3: Calcul du pérmiètre d'un cercle.

```
// perimetre  -  11.03.2014

// ****************************************
// Calcul du périmètre d'un cercle
// ****************************************

VARIABLES
  rayon EST_DU_TYPE NOMBRE
  perimetre EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  // lecture du rayon
  LIRE rayon
  AFFICHER "rayon = "
  AFFICHER rayon
  // calcul du périmètre
  perimetre PREND_LA_VALEUR 2 * Math.PI * rayon
  // affichage du résutlat
  AFFICHER "périmètre = "
  AFFICHER perimetre
FIN_ALGORITHME
```

2.2 Faire une condition

Algorithme 2.4: Affiche le nombre le plus grand.

```
// le_plus_grand  -  11.03.2014

// ****************************************
// Lire x et y et dire lequel est le plus grand
// ****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
  y EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //lecture de x et y
  LIRE x
  LIRE y
  AFFICHER "x = "
  AFFICHER x
  AFFICHER "y = "
  AFFICHER y
  //test entre x et y
  SI (x > y) ALORS
    DEBUT_SI
    AFFICHER "x est plus grand"
    FIN_SI
    SINON
      DEBUT_SINON
      SI (y > x) ALORS
        DEBUT_SI
        AFFICHER "y est plus grand"
        FIN_SI
        SINON
          DEBUT_SINON
          AFFICHER "x et y sont égaux"
          FIN_SINON
      FIN_SINON
FIN_ALGORITHME
```

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

```
// discriminent  -  11.03.2014

// ****************************************
// Calcul des racines réelles d'un polynome de degré 2
// ****************************************

VARIABLES
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
  c EST_DU_TYPE NOMBRE
  delta EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  AFFICHER "On va résoudre l'équation a*x^2 + b*x + c = 0"
  //Lecture des variables
  LIRE a
  LIRE b
  LIRE c
  //Calcul du discriminent
  delta PREND_LA_VALEUR b*b-4*a*c
  AFFICHER "Delta = "
  AFFICHER delta
  //Test du discriminent
  SI (delta >= 0) ALORS
    DEBUT_SI
    SI (delta == 0) ALORS
      DEBUT_SI
      AFFICHER "L'équation a une seule solution"
      AFFICHERCALCUL -b / (2 * a)
      FIN_SI
      SINON
        DEBUT_SINON
        AFFICHER "L'équation a deux solutions réelles"
        AFFICHERCALCUL (-b + sqrt(delta)) / (2 * a)
        AFFICHERCALCUL (-b - sqrt(delta)) / (2 * a)
        FIN_SINON
    FIN_SI
    SINON
      DEBUT_SINON
      AFFICHER "L'équation a deux solutions complexes"
      FIN_SINON
FIN_ALGORITHME
```

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

```
// racine_x  -  11.03.2014

// ****************************************
// Calcul de la racine carré de x si x est positif.
// ****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  // Lecture de x
  LIRE x
  AFFICHER "x = "
  AFFICHER x
  // test de la valeur de x et calcul de sqrt(x)
  SI ( x >= 0) ALORS
    DEBUT_SI
    AFFICHER "RACINE(x) = "
    AFFICHERCALCUL sqrt(x)
    FIN_SI
    SINON
      DEBUT_SINON
      AFFICHER "x est négatif"
      FIN_SINON
FIN_ALGORITHME
```

2.3 Répéter une action

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

```
calc_val_fonction_1  -  11.03.2014

****************************************
calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //boucle sur les valeurs de x
  POUR x ALLANT_DE -5 A 5
    DEBUT_POUR
    AFFICHERCALCUL x * x
    FIN_POUR
FIN_ALGORITHME
```

Algorithme 2.8: Illustration de l'utilisation d'une boucle TANT_QUE.

```
// calc_val_fonction_2  -  11.03.2014

// ****************************************
// calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
// ****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
  pas EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //initialisation
  x PREND_LA_VALEUR -5
  pas PREND_LA_VALEUR 0.5
  //boucle sur les valeurs de x
  TANT_QUE (x <= 5) FAIRE
    DEBUT_TANT_QUE
    AFFICHERCALCUL x * x
    x PREND_LA_VALEUR x + pas
    FIN_TANT_QUE
FIN_ALGORITHME
```

2.4 Variable indicée

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

```
// liste_impair  -  11.03.2014

// ****************************************
// Remplissage et affichage d'une liste des 20 premiers entiers impmairs
// ****************************************

VARIABLES
  x EST_DU_TYPE LISTE
  i EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Nombre d'entier
  n PREND_LA_VALEUR 20
  //Remplissage de la liste
  POUR i ALLANT_DE 0 A n - 1
    DEBUT_POUR
    x[i] PREND_LA_VALEUR 2 * i + 1
    FIN_POUR
  //Affichage de la liste
  AFFICHER "Liste des 20 premiers nombres impaires"
  POUR i ALLANT_DE 0 A n - 1
    DEBUT_POUR
    AFFICHER x[i]
    FIN_POUR
FIN_ALGORITHME
```

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

```
// liste_random  -  11.03.2014

// ****************************************
// Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
  maListe EST_DU_TYPE LISTE
  x EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Choix du nombre de valeurs
  n PREND_LA_VALEUR 15
  //Remplissage et affichage de la liste
  POUR i ALLANT_DE 1 A n
    DEBUT_POUR
    x PREND_LA_VALEUR 2. * random() - 1.
    maListe[i] PREND_LA_VALEUR x
    AFFICHER x
    FIN_POUR
FIN_ALGORITHME
```

2.5 Sous programme

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

```
// calc_val_fonction_3  -  11.03.2014

// ****************************************
// calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
// ****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
  pas EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //initialisation
  x PREND_LA_VALEUR -5
  pas PREND_LA_VALEUR 0.5
  //boucle sur les valeurs de x
  TANT_QUE (x <= 5) FAIRE
    DEBUT_TANT_QUE
    AFFICHERCALCUL F1(x)
    x PREND_LA_VALEUR x + pas
    FIN_TANT_QUE
FIN_ALGORITHME

// Fonction numérique utilisée :
F1(x)=x*x
```

2.6 Premiers petits programmes

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

```
// factorielle  -  11.03.2014

// ****************************************
// Calcul de factorielle n
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
  factorielle EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //lecture de n
  LIRE n
  AFFICHER "Calcul de factorielle "
  AFFICHER n
  //initialisation
  factorielle PREND_LA_VALEUR 1
  // calcul de la factorielle
  POUR i ALLANT_DE 2 A n
    DEBUT_POUR
    factorielle PREND_LA_VALEUR factorielle * i
    FIN_POUR
  //affichage du résultat
  AFFICHER "Résultat = "
  AFFICHER factorielle
FIN_ALGORITHME
```

Algorithme 2.13: Calcul d'un produit.

```
// produit  -  11.03.2014

// ****************************************
// Calcul du produit des n premiers entiers impairs
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
  produit EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //lecture de n
  LIRE n
  AFFICHER "Calcul du produit des entiers impairs entre 1 et "
  AFFICHER n
  //initialisation
  produit PREND_LA_VALEUR 1
  POUR i ALLANT_DE 3 A n
    DEBUT_POUR
    SI (i % 2 == 1) ALORS
      DEBUT_SI
      produit PREND_LA_VALEUR produit * i
      FIN_SI
    FIN_POUR
  //affichage du résultat
  AFFICHER "Résultat = "
  AFFICHER produit
FIN_ALGORITHME
```

Algorithme 2.14: Calcul d'une somme.

```
// somme  -  11.03.2014

// ****************************************
// Calcul de la somme des n premiers entiers
// ****************************************

VARIABLES
  n EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
  somme EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Lecture de n
  LIRE n
  AFFICHER "Calcul de la somme des entiers de 1 à "
  AFFICHER n
  //initialisation de la somme
  somme PREND_LA_VALEUR 0
  POUR i ALLANT_DE 1 A n
    DEBUT_POUR
    somme PREND_LA_VALEUR somme + i
    FIN_POUR
  //affichage du résultat
  AFFICHER "Résultat = "
  AFFICHER somme
FIN_ALGORITHME
```

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

```
// population  -  11.03.2014

// ****************************************
// Une population est réduite de 40% tous les 3 ans.
// Au bout de combien d'années la population est négligeable
// (inférireure à 0.1% de la population initiale) ?
// ****************************************

VARIABLES
  population EST_DU_TYPE NOMBRE
  an EST_DU_TYPE NOMBRE
  perte EST_DU_TYPE NOMBRE
  seuil EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //initialisation
  population PREND_LA_VALEUR 100
  an PREND_LA_VALEUR 0
  perte PREND_LA_VALEUR 0.4
  seuil PREND_LA_VALEUR 0.1 / 100 * population
  //boucle sur les années
  TANT_QUE (population > seuil) FAIRE
    DEBUT_TANT_QUE
    an PREND_LA_VALEUR an + 3
    population PREND_LA_VALEUR population * (1 - perte)
    AFFICHER an
    AFFICHER "  "
    AFFICHER population
    FIN_TANT_QUE
  //résultats
  AFFICHER "Au bout de "
  AFFICHER an
  AFFICHER " ans, la population est inférieure à 0.1% de la population initiale."
FIN_ALGORITHME
```

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

```
// minmax  -  11.03.2014

// ****************************************
// Recherche du maximum et du minimum dans une liste pseudo-alétoire
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
  maListe EST_DU_TYPE LISTE
  mini EST_DU_TYPE NOMBRE
  maxi EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //nombre de points
  LIRE n
  AFFICHER "n = "
  AFFICHER n
  //Remplissage d'une liste pseudo-aléatoire de nombres entre 0 et 100
  POUR i ALLANT_DE 1 A n
    DEBUT_POUR
    maListe[i] PREND_LA_VALEUR 100. * random()
    AFFICHER maListe[i]
    FIN_POUR
  //Recherche du minimum et du maximum
  //initialisation
  mini PREND_LA_VALEUR maListe[1]
  maxi PREND_LA_VALEUR maListe[1]
  POUR i ALLANT_DE 1 A n
    DEBUT_POUR
    //Recherche du minimum
    SI (maListe[i] < mini) ALORS
      DEBUT_SI
      mini PREND_LA_VALEUR maListe[i]
      FIN_SI
    //Recherche maximum
    SI (maListe[i] > maxi) ALORS
      DEBUT_SI
      maxi PREND_LA_VALEUR maListe[i]
      FIN_SI
    FIN_POUR
  AFFICHER "maximum = "
  AFFICHER maxi
  AFFICHER "minimum = "
  AFFICHER mini
FIN_ALGORITHME
```

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

```
// marche_aleatoire  -  11.03.2014

// ****************************************
// Marche aléatoire dans une plan 2D
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  npas EST_DU_TYPE NOMBRE
  amplitude EST_DU_TYPE NOMBRE
  x EST_DU_TYPE LISTE
  y EST_DU_TYPE LISTE
  wx EST_DU_TYPE NOMBRE
  wy EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //paramètres de la marche
  LIRE npas
  LIRE amplitude
  //initialisation
  x[1] PREND_LA_VALEUR 0
  y[1] PREND_LA_VALEUR 0
  //marche aléatoire
  POUR i ALLANT_DE 2 A npas
    DEBUT_POUR
    wx PREND_LA_VALEUR 2. * random() - 1.
    wy PREND_LA_VALEUR 2. * random() - 1.
    x[i] PREND_LA_VALEUR x[i - 1] + amplitude * wx
    y[i] PREND_LA_VALEUR y[i - 1] + amplitude * wy
    FIN_POUR
  //représentation de la marche
  POUR i ALLANT_DE 2 A npas
    DEBUT_POUR
    TRACER_SEGMENT (x[i - 1],y[i - 1])->(x[i],y[i])
    FIN_POUR
FIN_ALGORITHME
```

Algorithme 2.18: Calcul du nombre π par la méthode Monte Carlo.

```
// pi  -  14.03.2014

// ****************************************
// Calcul du nombre pi par Monte Carlo.
// Principe : La probabilité qu'un point de coordonnées (x,y)
// avec x et y dans l'intervalle [0;1] soit dans le quart de
// cercle de centre (0,0) et de rayon 1 est égal au rapport des
/// aires du quart de cercle de rayon 1 et du carré de largeur 1
// soit pi/4.
****************************************

VARIABLES
  x EST_DU_TYPE NOMBRE
  y EST_DU_TYPE NOMBRE
  n EST_DU_TYPE NOMBRE
  ntirage EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //Lecture du nombre de tirage
  LIRE ntirage
  //initialisation
  n PREND_LA_VALEUR 0
  //calcul du nombre pi
  POUR i ALLANT_DE 1 A ntirage
    DEBUT_POUR
    //tirage d'un point dans le carré de coté 1
    x PREND_LA_VALEUR random()
    y PREND_LA_VALEUR random()
    SI (x*x + y*y < 1) ALORS
      DEBUT_SI
      n PREND_LA_VALEUR n + 1
      TRACER_POINT (x,y)
      FIN_SI
    FIN_POUR
  //Affichage des résultats
  AFFICHER "pi = "
  AFFICHERCALCUL 4.0 * n / ntirage
  AFFICHER "% d'erreur = "
  AFFICHERCALCUL (Math.PI - 4.0 * n / ntirage) / Math.PI * 100.
FIN_ALGORITHME
```

Algorithme 2.19: Intégration par la méthode des trapèzes.

```
// trapeze  -  11.03.2014

// ********************************************************************
// Intégration par la méthode des trapèzes. L'intégration analytique
// de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
// application numérique -2.525369
//*********************************************************************

VARIABLES
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
  i EST_DU_TYPE NOMBRE
  npas EST_DU_TYPE NOMBRE
  integrale EST_DU_TYPE NOMBRE
  analytique EST_DU_TYPE NOMBRE
  pas EST_DU_TYPE NOMBRE
  x EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //valeur exacte
  analytique PREND_LA_VALEUR -2.525369
  //Lecture de l'intervalle
  LIRE a
  LIRE b
  //Nombre de segments
  LIRE npas
  //initialisation
  integrale PREND_LA_VALEUR 0
  pas PREND_LA_VALEUR (b - a) / npas
  //calcul de l'integrale
  POUR i ALLANT_DE 1 A npas
    DEBUT_POUR
    x PREND_LA_VALEUR a + (i - 1) * pas
    integrale PREND_LA_VALEUR integrale + pas * (F1(x) + F1(x + pas)) / 2
    FIN_POUR
  //résultat
  AFFICHER "Résultat = "
  AFFICHER integrale
  AFFICHER "Résidu = "
  AFFICHERCALCUL analytique - integrale
  AFFICHER "Précision = "
  AFFICHERCALCUL (analytique - integrale) / analytique * 100
FIN_ALGORITHME

// Fonction numérique utilisée :
F1(x)=(x * x - 3 * x - 6) * exp(-x)
```

Algorithme 2.20: Intégration par la méthode de Simpson.

```
// simpson  -  11.03.2014

// ************************************************************
// Intégration par la méthode de Simpson. L'intégration analytique
// de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
// application numérique -2.525369
// ************************************************************

VARIABLES
  a EST_DU_TYPE NOMBRE
  b EST_DU_TYPE NOMBRE
  npas EST_DU_TYPE NOMBRE
  integrale EST_DU_TYPE NOMBRE
  analytique EST_DU_TYPE NOMBRE
  pas EST_DU_TYPE NOMBRE
  x EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //valeur exacte
  analytique PREND_LA_VALEUR -2.525369
  //Lecture de l'intervalle
  LIRE a
  LIRE b
  //Nombre de segments
  LIRE npas
  //initialisation
  integrale PREND_LA_VALEUR 0
  pas PREND_LA_VALEUR (b - a) / npas
  x PREND_LA_VALEUR a
  //calcul de l'integrale
  TANT_QUE (x < b) FAIRE
    DEBUT_TANT_QUE
    integrale PREND_LA_VALEUR integrale + pas / 3 * (F1(x) + 4 * F1(x + pas) + F1(x + 2 * pas))
    x PREND_LA_VALEUR x + 2 * pas
    FIN_TANT_QUE
  //résultat
  AFFICHER "Résultat = "
  AFFICHER integrale
  AFFICHER "Résidu = "
  AFFICHERCALCUL analytique - integrale
  AFFICHER "Précision = "
  AFFICHERCALCUL (analytique - integrale) / analytique * 100
FIN_ALGORITHME

// Fonction numérique utilisée :
F1(x)=(x * x - 3 * x - 6) * exp(-x)
```

Algorithme 2.21: Procédé d'orthogonalisation de Gramm-Schmidt.

```
// schmidt  -  13.03.2014

// ****************************************
// Procédé d'orthogonalisation de Gram-Schimdt.
// Soit u et v deux vecteurs. On cherche le vecteur vp le plus proche de v orthogonal à u.
// ****************************************

VARIABLES
  i EST_DU_TYPE NOMBRE
  u EST_DU_TYPE LISTE
  v EST_DU_TYPE LISTE
  vp EST_DU_TYPE LISTE
  scalaire EST_DU_TYPE NOMBRE
  normu EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  //vecteur u
  u[1] PREND_LA_VALEUR 1
  u[2] PREND_LA_VALEUR 0
  u[3] PREND_LA_VALEUR 0
  normu PREND_LA_VALEUR sqrt(u[1]*u[1] + u[2]*u[2] + u[3]*u[3])
  //vecteur v
  v[1] PREND_LA_VALEUR 1
  v[2] PREND_LA_VALEUR 2
  v[3] PREND_LA_VALEUR 3
  //calcul du produit scalaire
  scalaire PREND_LA_VALEUR u[1] * v[1] + u[2] * v[2] + u[3] * v[3]
  AFFICHER "u.v = "
  AFFICHER scalaire
  //orthogonalisation de schmidt
  SI (scalaire != 0) ALORS
    DEBUT_SI
    POUR i ALLANT_DE 1 A 3
      DEBUT_POUR
      vp[i] PREND_LA_VALEUR v[i] - scalaire / normu * u[i]
      FIN_POUR
    //verification
    AFFICHER "u.vp = "
    AFFICHERCALCUL u[1] * vp[1] + u[2] * vp[2] + u[3] * vp[3]
    AFFICHER "vp = "
    POUR i ALLANT_DE 1 A 3
      DEBUT_POUR
      AFFICHER vp[i]
      AFFICHER "  "
      FIN_POUR
    FIN_SI
FIN_ALGORITHME
```

Programming from examples

Chapitre 3

Programmes en Fortran

3.1 Lire – écrire – compter

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

```
! aire_rectangle  -  11.03.2014

! ****************************************
! Calcul de l'aire d'un rectangle
! ****************************************

program aire_rectangle

    implicit none

    ! declaration
    real :: largeur, longueur

    ! Lecture des variables
    write(*,*) "Entrer la largeur"
    read(*,*) largeur
    write(*,*) "largeur = ", largeur

    write(*,*) "Entrer la longueur"
    read(*,*) longueur
    write(*,*) "longueur = ", longueur

    ! affiche le résultat
    write(*,*) "Aire du rectangle = ", largeur * longueur

end program aire_rectangle
```

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

```
! reste  -  11.03.2014

! ****************************************
! Calcul du reste de la division de deux entiers
! ****************************************

program prog_reste

    implicit none

    integer :: dividende, diviseur, reste

    ! lecture du dividende
    write(*,*) "Dividende = "
    read(*,*) dividende
    write(*,*) "Dividende = ", dividende

    ! lecture du diviseur
    write(*,*) "Diviseur = "
    read(*,*) diviseur
    write(*,*) "Diviseur = ", diviseur

    ! calcul du reste
    reste = mod(dividende, diviseur)

    ! affichage du resultat
    write(*,*) "Reste = ", reste

end program prog_reste
```

Code FORTRAN 3.3: Calcul du pérmiètre d'un cercle.

```
! perimetre  -  11.03.2014

! ****************************************
! Calcul du périmètre d'un cercle
! ****************************************

program prog_perimetre

    implicit none

    ! parametres
    double precision, parameter :: pi = 3.141592653

    ! variables
    double precision :: rayon, perimetre

    ! lecture du rayon
    write(*, *) "Rayon = "
    read(*, *) rayon
    write(*, *) "Rayon = ", rayon

    ! calcule le perimetre
    perimetre = 2. * pi * rayon
    write(*, *) "Perimetre = ", perimetre

end program prog_perimetre
```

3.2 Faire une condition

Code FORTRAN 3.4: Affiche le nombre le plus grand.

```
! le_plus_grand  -  11.03.2014

! ****************************************
! Lire x et y et dire lequel est le plus grand
! ****************************************

program le_plus_grand

    implicit none

    real :: x, y

    ! lecture de x et y
    write(*, *) "Entrer x : "
    read(*, *) x
    write(*, *) "x = ", x

    write(*, *) "Entrer y : "
    read(*, *) y
    write(*, *) "y = ", y

    ! test entre x et y
    if (x > y) then
        write(*, *) "x est plus grand"
    else if (x < y) then
        write(*, *) "y est plus grand"
    else
        write(*, *) "x et y sont égaux"
    end if
end program le_plus_grand
```

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

```
! discriminent  -  11.03.2014

! ****************************************
! Calcul des racines réelles d'un polynome de degré 2
! ****************************************

program discriminent

    implicit none

    ! déclaration
    real :: a, b, c, delta

    write(*, *) "On va résoudre l'équation a*x^2 + b*x + c = 0"

    ! Lecture des variables
    write(*, *) "Entrer a"
    read(*, *) a
    write(*, *) "a = ", a

    write(*, *) "Entrer b"
    read(*, *) b
    write(*, *) "b = ", b

    write(*, *) "Entrer c"
    read(*, *) c
    write(*, *) "c = ", c

    ! Calcul du discriminent
    delta = b**2 - 4 * a * c
    write(*, *) "Delta = ", delta

    ! Test du discriminent
    if (delta > 0.) then
        write(*, *) "L'équation a deux solutions réelles :"
        write(*, *) (-b + sqrt(delta)) / (2.0 * a)
        write(*, *) (-b - sqrt(delta)) / (2.0 * a)
    else if (delta < 0.) then
        write(*, *) "L'équation a deux solutions complexes"
    else
        write(*, *) "L'équatuion a une seule solution :"
        write(*, *) -b / (2.0 * a)
    end if

end program discriminent
```

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

```
! racine_x  -  11.03.2014

! ****************************************
! Calcul de la racine carré de x si x est positif.
! ****************************************

program racine_x

    implicit none

    real :: x

    ! lecture de x
    write(*, *) "x = "
    read(*, *) x
    write(*, *) "x = ", x

    ! test de la valeur de x et calcul de sqrt(x)
    if (x >= 0) then
        write(*, *) "racine(x) = ", sqrt(x)
    else
        write(*, *) "x est négatif"
    end if

end program racine_x
```

3.3 Répéter une action

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

```
! calc_val_fonction_1  -  11.03.2014

! ****************************************
! calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
! ****************************************

program calc_val_fonction_1

    implicit none

    ! declaration
    integer :: x

    ! boucle sur les valeurs de x
    do x = -5, 5, 1
        write(*, *) x, x**2
    end do

end program calc_val_fonction_1
```

Code FORTRAN 3.8: Illustration de l'utilisation d'une boucle TANT_QUE.

```
! calc_val_fonction_2  -  11.03.2014

! ****************************************
! calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
! ****************************************

program calc_val_fonction_2

    implicit none

    ! declaration
    real :: x, pas

    ! initialisation
    x = -5.0
    pas = 0.5

    ! boucle sur les valeurs de x
    do while (x <= 5.0)
        write(*, *) x, x**2
        x = x + pas
    end do

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).

```
! liste_impair  -  11.03.2014

! ****************************************
! Remplissage et affichage d'une liste des 20 premiers entiers impmairs
! tableau statique
! ****************************************

program liste_impair

    implicit none

    integer                 :: i
    ! nombre de valeurs
    integer, parameter      :: n = 20
    ! declaration d'un tableau statique
    integer, dimension(1:n) :: x

    ! Remplissage de la liste
    do i = 1, n
        x(i) = 2 * i - 1
    end do

    ! affichage de la liste
    write(*, *) "Liste des ", n, " premiers nombres impaires"
    do i = 1, n
        write(*, *) i, x(i)
    end do

    ! affichage 10 par ligne
    write(*, "(10i5)") (x(i), i = 1, n)

end program liste_impair
```

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

```
! liste_random  -  11.03.2014

! ****************************************
! Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
! tableau statique
! ****************************************

program liste_random

    implicit none

    integer              :: i
    real                 :: x
    integer, parameter   :: n = 15
    real, dimension(1:n) :: maListe

    ! Remplissage et affichage de la liste
    do i = 1, n
        call random_number(x)
        maListe(i) = 2.0 * x - 1.0
        write(*, *) i, maListe(i)
    end do

    ! plus direct
    call random_number(maListe(:))
    maListe(:) = 2.0 * maListe(:) - 1.0

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).

```
! liste_impair  -  11.03.2014

! ****************************************
! Remplissage et affichage d'une liste des 20 premiers entiers impmairs
! tableau dynamique
! ****************************************

program liste_impair

    implicit none

    integer :: i, n
    ! declaration d'un tableau dynamique
    integer, dimension(:), allocatable :: x

    ! lecture du nombre de valeurs
    write(*, *) "Nombre de nombres impairs :"
    read(*, *) n
    write(*, *) "n = ", n

    ! allocation de la mémoire pour le tableau
    allocate(x(1:n))

    ! Remplissage de la liste
    do i = 1, n
        x(i) = 2 * i - 1
    end do

    ! affichage 10 par ligne
    write(*, "(10i5)") (x(i), i = 1, n)

    ! libération de la memoire
    deallocate(x)

end program liste_impair
```

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

```
! liste_random  -  11.03.2014

! ****************************************
! Remplissage et affichage d'une liste aléatoire de nombre entre -1 et 1
! tableau dynamique
! ****************************************

program liste_random

    implicit none

    integer                          :: i, n
    real, dimension(:), allocatable :: maListe

    ! nombre de valeurs
    write(*, *) "Nombres de valeurs : "
    read(*, *) n
    write(*, *) "n = ", n

    ! réservation mémoire
    allocate(maListe(1:n))

    ! Remplissage de la liste
    call random_number(maListe(:))
    maListe(:) = 2.0 * maListe(:) - 1.0

    ! affichage des nombres
    do i = 1, n
        write(*, *) i, maListe(i)
    end do

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.

```
! calc_val_fonction_3  -  11.03.2014

! ****************************************
! calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5]
! ****************************************

program calc_val_fonction_3

    implicit none

    ! déclaration
    real :: x, pas

    ! initialisation
    x = -5.0
    pas = 0.5

    ! boucle sur les valeurs de x
    do while (x <= 5)
        write(*, *) x, f(x)
        x = x + pas
    end do

    contains

    ! fonction numerique
    real function f(x)

        implicit none

        real :: x

        f = x**2

    end function f

end program calc_val_fonction_3
```

3.6 Premiers petits programmes

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

```
! factorielle  -  11.03.2014

! ****************************************
! Calcul de factorielle n
! ****************************************

program factorielle

    implicit none

    integer :: i, n, facto

    ! lecture de n
    write(*, *) "Entrer n :"
    read(*, *) n
    write(*, *) "n = ", n

    ! initialiation
    facto = 1

    ! calcul de la factorielle
    do i = 2, n
        facto = facto * i
    end do

    ! affichage résultat
    write(*, *) "Résultat = ", facto

end program factorielle
```

Code FORTRAN 3.15: Calcul d'un produit.

```
! produit  -  11.03.2014

! ****************************************
! Calcul du produit des n premiers entiers impairs
! ****************************************

program prog_produit

    implicit none

    integer :: i, n, produit

    ! lecture de n
    write(*, *) "Entrer n : "
    read(*,*) n
    write(*, *) "Calcul du produit des premiers entiers impairs entre 1 et ", n

    ! initialisation
    produit = 1

    ! boucle
    do i = 3, n, 1
        if (mod(i, 2) == 1) then
            produit = produit * i
        end if
    end do
    write(*, *) "produit = ", produit

    ! autre boucle
    produit = 1
    do i = 3, n, 2
        produit = produit * i
    end do
    write(*, *) "produit = ", produit

end program prog_produit
```

Code FORTRAN 3.16: Calcul d'une somme.

```
! somme  -  11.03.2014

! ****************************************
! Calcul de la somme des n premiers entiers
! ****************************************

program prog_somme

    implicit none

    integer :: i, n, somme

    ! lecture de n
    write(*, *) "n ="
    read(*,*) n
    write(*, *) "n = ", n

    ! initialisation
    somme = 0

    ! boucle
    do i = 1, n, 1
        somme = somme + i
    end do

    ! résultat
    write(*, *) "somme = ", somme

end program prog_somme
```

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

```
! population  -  11.03.2014

! ****************************************
! Une population est réduite de 40% tous les 3 ans.
! Au bout de combien d'années la population est négligeable
! (inférireure à 0.1% de la population initiale) ?
! ****************************************

program prog_population

    implicit none

    integer :: an
    real    :: population, seuil, perte

    ! initialisation
    population = 100.0
    an = 0
    write(*, *) an, population

    ! paramètres
    perte = 0.4
    seuil = 0.1 / 100. * population

    ! boucle sur les années
    do while (population > seuil)
        an = an + 3
        population = population * (1.0 - perte)
        write(*, *) an, population
    end do

    ! résultats
    write(*, *) "Au bout de ", an, " ans, la population est inférieure à 0.1% de la population initi

end program prog_population
```

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

```
! minmax  -  11.03.2014

! ****************************************
! Recherche du maximum et du minimum dans une liste pseudo-alétoire
! ****************************************

program minimax

    implicit none

    integer, parameter   :: n = 35
    integer              :: i
    real                 :: mini, maxi
    real, dimension(1:n) :: maListe

    ! Remplissage d'une liste pseudo-aléatoire de nombres entre 0 et 100
    call random_number(maListe(:))
    maListe(:) = 100.0 * maListe(:)
    write(*, "(10F8.3)") (maListe(i), i = 1, n)

    ! initialisation
    mini = maListe(1)
    maxi = maListe(1)

    ! recherche du maximum et du minimum
    do i = 1, n
        ! recherche minimum
        if (maListe(i) < mini) then
            mini = maListe(i)
        end if

        ! recherche maximum
        if (maListe(i) > maxi) then
            maxi = maListe(i)
        end if
    end do

    ! résultats
    write(*, *) "Résultats :"
    write(*, *) "maximum = ", maxi
    write(*, *) "minimum = ", mini

    ! avec les fonctions internes min(), max()
    mini = maListe(1)
    maxi = maListe(1)

    do i = 1, n
        mini = min(mini, maListe(i))
        maxi = max(maxi, maListe(i))
    end do

    write(*, *) "maximum = ", maxi
    write(*, *) "minimum = ", mini

    ! avec les fonctions internes maxval(), minval()
    mini = minval(maListe(:))
    maxi = maxval(maListe(:))
```

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

```
! marche_aleatoire  -  11.03.2014

! ****************************************
! Marche aléatoire dans une plan 2D
! ****************************************

program marche_aleatoire

    implicit none

    integer, parameter                :: d = 2      ! dimension
    integer                           :: i, k, npas
    real                              :: amplitude
    real, dimension(d)                :: w
    real, dimension(:,:), allocatable :: x

    ! paramètres de la marche
    write(*, *) "Nombre de pas : "
    read(*, *) npas
    write(*, *) "npas = ", npas

    write(*, *) "Amplitude des déplacement : "
    read(*, *) amplitude
    write(*, *) "amplitude = ", amplitude

    ! allocation memoire
    allocate(x(npas, d))

    ! initialisation
    do k = 1, d
        x(1, k) = 0.0
    end do

    ! marche aleatoire
    do i = 2, npas
        do k = 1, d
            call random_number(w(k))
            w(k) = 2.0 * w(k) - 1.0
            x(i, k) = x(i - 1, k) + amplitude * w(k)
        end do
    end do

    ! enregistrement de la trajectoire dans un fichier
    open(unit = 10, file = "traj.dat", action = "write")
    do i = 1, npas
        write(10, *) (x(i, k), k = 1, d)
    end do
    close(unit = 10)

end program marche_aleatoire
```

Code FORTRAN 3.20: Calcul du nombre π par la méthode Monte Carlo.

```
program prog_pi

  implicit none

  integer                     :: i, n, ntirage
  double precision            :: x, y
  double precision, parameter :: pi = 3.141592653589793d0

  ! lecture du nombre de tirage
  write(*, *) "Nombre de tirage : "
  read(*, *) ntirage
  write(*, *) "ntirage = ", ntirage

  ! initialisation
  n = 0

  ! calcul du nombre pi
  do i = 1, ntirage, 1
      call random_number(x)
      call random_number(y)

      if (x**2 + y**2 < 1.d0) then
          n = n + 1
      end if
  end do

  write(*, *) "pi = ", 4.0 * dble(n) / dble(ntirage)
  write(*, *) "% d'erreur = ", (pi - 4.d0 * dble(n) / dble(ntirage)) / pi * 100.d0

end program prog_pi
```

Code FORTRAN 3.21: Intégration par la méthode des trapèzes.

```
! trapeze  -  11.03.2014

! ****************************************************************
! Intégration par la méthode des trapèzes. L'intégration analytique
! de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
! application numérique -2.525369
!*****************************************************************

program trapeze

    implicit none

    integer         :: i, npas
    real            :: a, b, pas, integrale, x

    ! valeur exacte
    real, parameter :: analytique = -2.525369

    ! Lecture de l'intervalle
    write(*, *) "a = "
    read(*, *) a
    write(*, *) "a = ", a

    write(*, *) "b = "
    read(*, *) b
    write(*, *) "b = ", b

    ! Nombre de segments
    write(*, *) "npas = "
    read(*, *) npas
    write(*, *) "npas = ", npas

    ! initialisation
    integrale = 0.0
    pas = (b - a) / real(npas)

    ! calcul de l'integrale
    do i = 1, npas
        x = a + (i - 1) * pas
        integrale = integrale + pas * (f(x) + f(x + pas)) / 2.0
    end do

    ! résultats
    write(*, *) "Résultat  = ", integrale
    write(*, *) "Résidu    = ", analytique - integrale
    write(*, *) "Précision = ", (analytique - integrale) / analytique * 100.0

    contains

        ! fonction numérique utilisée
        real function f(x)
            implicit none
            real :: x
            f = (x**2 - 3 * x - 6) * exp(-x)
       end function f

end program trapeze
```

Code FORTRAN 3.22: Intégration par la méthode de Simpson.

```
! simpson  -  11.03.2014

! ************************************************************
! Intégration par la méthode de Simpson. L'intégration analytique
! de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
! application numérique -2.525369
! ************************************************************

program simpson

    implicit none

    integer         :: npas
    real            :: a, b, pas, integrale, x

    ! valeur exacte
    real, parameter :: analytique = -2.525369

    ! Lecture de l'intervalle
    write(*, *) "a = "
    read(*, *) a
    write(*, *) "a = ", a

    write(*, *) "b = "
    read(*, *) b
    write(*, *) "b = ", b

    ! Nombre de segments
    write(*, *) "npas = "
    read(*, *) npas
    write(*, *) "npas = ", npas

    ! initialisation
    integrale = 0.0
    pas = (b - a) / real(npas)
    x = a

    ! calcul de l'integrale
    do while (x < b)
        integrale = integrale + pas / 3.0 * (f(x) + 4.0 * f(x + pas) + f(x + 2.0 * pas))
        x = x + 2.0 * pas
    end do

    ! résultats
    write(*, *) "Résultat  = ", integrale
    write(*, *) "Résidu    = ", analytique - integrale
    write(*, *) "Précision = ", (analytique - integrale) / analytique * 100.0

    contains

        ! fonction numérique utilisée
       real function f(x)
           implicit none
           real :: x
           f = (x**2 - 3 * x - 6) * exp(-x)
       end function f

```

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

```
! schmidt  -  13.03.2014

! ****************************************
! Procédé d'orthogonalisation de Gram-Schimdt.
! Soit u et v deux vecteurs. On cherche le vecteur vp
! le plus proche de v orthogonal à u.
! ****************************************

program schmidt

    implicit none

    integer             :: i
    real                :: scalaire, normu
    real, dimension(3) :: u, v, vp

    ! vecteur u
    u(1) = 1.0
    u(2) = 0.0
    u(3) = 0.0
    normu = sqrt(sum((u(:))**2))

    ! vecteur v
    v(1) = 1.0
    v(2) = 2.0
    v(3) = 3.0

    ! calcul du produit scalaire
    scalaire = dot_product(u, v)
    write(*, *) "u.v = ", scalaire

    ! orthogonalisation de schimdt
    if (scalaire /= 0) then
        vp(:) = v(:) - scalaire / normu * u(:)
        ! verification
        write(*, *) "u.vp = ", dot_product(u, vp)
        write(*, *) "vp = ", (vp(i), i = 1, 3)
    end if

end program schmidt
```

Programming from examples

Chapitre 4

Programmes en Python

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" Présentation du code """

def fonction(argument1, argument2):
    """ description de la fonction """

    # programmation de la fonction

    return "une valeur"

if __name__ == "__main__":
    # instructions à exécuter, par exemple
    #    * lecture des parametres
    #    * appel de la fonction
    fonction()
```

4.1 Lire – écrire – compter

4.1.1 Versions sans fonction

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" aire_rectangle  -  11.03.2014

Calcul de l'aire d'un rectangle """

# lecture des variables
largeur = float(raw_input("largeur = "))
print("largeur = {0}".format(largeur))

longueur = float(raw_input("largeur = "))
print("longueur = {0}".format(longueur))

# arriche le résultats
print("Aire du rectangle = {0} ".format(largeur * longueur))
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" reste  -  11.03.2014

Calcul du reste de la division de deux entiers """

# lecture des valeurs
dividende = int(raw_input("entrer le dividende : "))
print("dividende = {0}".format(dividende))

diviseur = int(raw_input("entre le diviseur : "))
print("diviseur = {0}".format(diviseur))

# calcul du reste
reste = dividende % diviseur

# affichage du résultat
print("reste = {0}".format(reste))
```

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" perimetre  -  11.03.2014

Calcul du périmètre d'un cercle """

import math

# lecture du rayon
rayon = float(raw_input("entrer le rayon : "))
print("rayon = {0}".format(rayon))

# calcul du perimètre
perimetre = 2.0 * math.pi * rayon

# affichage du résultat
print("périmètre = {0}".format(perimetre))
```

4.1.2 Versions avec une fonction

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" aire_rectangle  -  11.03.2014 """

def aire_rectangle(largeur, longueur):
    """ Calcul de l'aire d'un rectangle """
    return largeur * longueur

if __name__ == "__main__":
    # lecture des variables
    largeur = float(raw_input("largeur = "))
    print("largeur = {0}".format(largeur))

    longueur = float(raw_input("largeur = "))
    print("longueur = {0}".format(longueur))

    # arriche le résultats
    print("Aire du rectangle = {0} ".format(aire_rectangle(largeur, longueur)))
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" reste  -  11.03.2014 """

def prog_reste(dividende, diviseur):
    """ Calcul du reste de la division de deux entiers """

    reste = dividende % diviseur

    print("reste = {0}".format(reste))

if __name__ == "__main__":
    # lecture des valeurs
    dividende = int(raw_input("entrer le dividende : "))
    print("dividende = {0}".format(dividende))

    diviseur = int(raw_input("entre le diviseur : "))
    print("diviseur = {0}".format(diviseur))

    prog_reste(dividende, diviseur)
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" perimetre  -  11.03.2014 """

import math

def perimetre(rayon):
    """ Calcul du périmètre d'un cercle """
    return 2.0 * math.pi * rayon

if __name__ == "__main__":
    # lecture du rayon
    rayon = float(raw_input("entrer le rayon : "))
    print("rayon = {0}".format(rayon))

    # affichage du résultat
    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.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" le_plus_grand  -  11.03.2014

Lire x et y et dire lequel est le plus grand """

# lecture de x et y
x = float(raw_input("entrer x : "))
print("x = {0}".format(x))

y = float(raw_input("entrer y : "))
print("y = {0}".format(y))

# test entre x et y
if x > y:
    print("x est plus grand")
    print("le plus grand = {0}".format(x))
elif y > x:
    print("y est plus grand")
    print("le plus grand = {0}".format(y))
else:
    print("x et y sont égaux")
    print("x = {0}\t y = {1}".format(x, y))
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" discriminent  -  11.03.2014

Calcul des racines réelles d'un polynome de degré 2 """

from math import sqrt

print("On va résoudre l'équation a*x^2 + b*x + c = 0")

# lecture des variables
a = float(raw_input("entrer a : "))
print("a = {0}".format(a))

b = float(raw_input("entrer b : "))
print("b = {0}".format(b))

c = float(raw_input("entrer c : "))
print("c = {0}".format(c))

# Calcul du discriminent
delta = b**2 - 4. * a * c
print("delta = {0}".format(delta))

# Test du discriminent
if delta > 0:
    print("L'équation a deux solutions")
    print("x1 = {0}".format((-b - sqrt(delta)) / (2. * a)))
    print("x2 = {0}".format((-b + sqrt(delta)) / (2. * a)))
elif delta < 0.:
    print("L'équation a deux solutions complexes")
else:
    print("L'équation a une seule solution")
    print("x  = {0}".format(-b / (2.0 * a)))
```

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" racine_x  -  11.03.2014

Calcul de la racine carré de x si x est positif. """

from math import sqrt

# lecture de x
x = float(raw_input("entrer x : "))
print("x = {0}".format(x))

# test de la valeur de x et calcul de sqrt(x)
if x >= 0:
    print("RACINE(x) = {0}".format(sqrt(x)))
else:
    print("x est négatif")
```

4.2.2 Versions avec une fonction

Code Python 4.11: Affiche le nombre le plus grand.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" le_plus_grand  -  11.03.2014 """

def le_plus_grand(x, y):
    """ donne le nombre le plus grand """

    # test entre x et y
    if x > y:
        print("x est plus grand")
        print("le plus grand = {0}".format(x))
    elif y > x:
        print("y est plus grand")
        print("le plus grand = {0}".format(y))
    else:
        print("x et y sont égaux")
        print("x = {0}\t y = {1}".format(x, y))

if __name__ == "__main__":
    # lecture de x et y
    x = float(raw_input("entrer x : "))
    print("x = {0}".format(x))

    y = float(raw_input("entrer y : "))
    print("y = {0}".format(y))

    # appel de la fonction
    le_plus_grand(x, y)
```

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" discriminent  -  11.03.2014 """

from math import sqrt

def discriminent(a, b, c):
    """ Calcul des racines réelles d'un polynome de degré 2 """

    # Calcul du discriminent
    delta = b**2 - 4. * a * c
    print("delta = {0}".format(delta))

    # Test du discriminent
    if delta > 0:
        print("L'équation a deux solutions")
        print("x1 = {0}".format((-b - sqrt(delta)) / (2. * a)))
        print("x2 = {0}".format((-b + sqrt(delta)) / (2. * a)))
    elif delta < 0.:
        print("L'équation a deux solutions complexes")
    else:
        print("L'équation a une seule solution")
        print("x  = {0}".format(-b / (2.0 * a)))

if __name__ == "__main__":
    print("On va résoudre l'équation a*x^2 + b*x + c = 0")

    # lecture des variables
    a = float(raw_input("entrer a : "))
    print("a = {0}".format(a))

    b = float(raw_input("entrer b : "))
    print("b = {0}".format(b))

    c = float(raw_input("entrer c : "))
    print("c = {0}".format(c))

    discriminent(a, b, c)
```

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" racine_x  -  11.03.2014 """

from math import sqrt

def racine(x):
    """ Calcul de la racine carré de x si x est positif. """

    if x >= 0:
        print("RACINE(x) = {0}".format(sqrt(x)))
    else:
        print("x est négatif")

if __name__ == "__main__":
    # lecture de x
    x = float(raw_input("entrer x : "))
    print("x = {0}".format(x))

    racine(x)
```

4.3 Répéter une action

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" calc_val_fonction_1  -  11.03.2014 """

def calc_val_fonction_1():
    """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """

    for x in range(-5, 6, 1):
        print(x**2)

if __name__ == "__main__":
    calc_val_fonction_1()
```

Code Python 4.15: Illustration de l'utilisation d'une boucle TANT_QUE.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" calc_val_fonction_2  -  11.03.2014 """

def calc_val_fonction_2():
    """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """

    # initialisation
    x = -5.0
    pas = 0.5

    # boucle sur les valeurs de x
    while x <= 5:
        print(x**2)
        x += pas

if __name__ == "__main__":
    calc_val_fonction_2()
```

4.4 Variable indicée

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" liste_impair  -  11.03.2014 """

def liste_impair(n = 20):
    """ Remplissage et affichage d'une liste des n premiers
        entiers impmairs """

    # creation d'une liste
    x = list()

    # remplissage de la liste
    for i in range(n):
        x.append(2 * i + 1)

    # ou
    x = [2 * i + 1 for i in range(n)]

    # affichage de la liste
    print("liste des {0} premiers nombres imapirs".format(n))
    for i in range(n):
        print(x[i])

if __name__ == "__main__":
    # lecture du nombre d'entiers
    n = int(raw_input("entrer n : "))
    print("n = {0}".format(n))

    liste_impair(n)
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" liste_random  -  11.03.2014 """

from random import random

def liste_random(n = 20):
    """ Remplissage et affichage d'une liste aléatoire
        de nombre entre -1 et 1 """

    # creation de la liste
    maListe = list()

    # remplissage et affichage de la liste
    for i in range(n):
        x = 2. * random() - 1.
        maListe.append(x)
        print(x)

    # ou
    maListe = [2. * random() - 1. for i in range(n)]

if __name__ == "__main__":
    # lecture du nombre de tirage aleatoire
    n = int(raw_input("entrer n : "))
    print("n = {0}".format(n))

    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.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" calc_val_fonction_3  -  11.03.2014 """

def calc_val_fonction_3(pas = 0.1):
    """ calcule des valeurs de la fonction f(x) = x^2 pour x dans [-5 ; 5] """

    # initialisation
    x = -5.0

    # boucle sur les valeurs de x
    while x <= 5:
        print(f(x))
        x += pas

def f(x):
    """ fonction numérique utilisée """
    return x**2

if __name__ == "__main__":
    pas = 0.5
    calc_val_fonction_3(pas)
    #calc_val_fonction_3()
```

4.6 Premiers petits programmes

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" factorielle  -  11.03.2014 """

def factorielle(n):
    """ Calcul de factorielle n """

    if n == 0:
        factorielle = 1
    else:
        # initialisation
        factorielle = 1

        # calcul
        for i in range(2, n + 1):
            factorielle *= i

    return factorielle

if __name__ == "__main__":
    # lecture de n
    n = int(raw_input("entrer n : "))
    print("n = {0}".format(n))

    # affichage du résultat
    print("{0}! = {1}".format(n, factorielle(n)))
```

Code Python 4.20: Calcul d'un produit.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" produit  -  11.03.2014 """

def prog_produit():
    """ Calcul du produit des n premiers entiers impairs """

    # lecture de n
    n = int(raw_input("entrer n : "))
    print("Calcul du produit des entiers impairs entre 1 et {0}".format(n))

    # initialisation
    produit = 1

    # calcul
    for i in range(3, n + 1):
        if i % 2 == 1:
            produit *= i

    # autre methode
    produit = 1
    for i in range(3, n + 1, 2):
        produit *= i

    # affichage du résultat
    print("Résultat = {0}".format(produit))

if __name__ == "__main__":
    prog_produit()
```

Code Python 4.21: Calcul d'une somme.

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" somme  -  11.03.2014 """

def prog_somme(n):
    """ Calcul de la somme des n premiers entiers """

    # initialisation
    somme = 0.0

    for i in range(n + 1):
         somme += i

    # ou somme = sum(range(n + 1))

    return somme

if __name__ == "__main__":
    # lecture de n
    n = int(raw_input("entrer n : "))
    print("n = {0}".format(n))

    print("Résultat = {0}".format(prog_somme(n)))
```

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

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" population  -  11.03.2014 """

def prog_population():
    """ Une population est réduite de 40% tous les 3 ans.
        Au bout de combien d'années la population est négligeable
        (inférireure à 0.1% de la population initiale) ? """

    # initialisation
    population = 100.0
    an = 0
    perte = .4
    seuil = .1 / 100. * population

    # boucle sur les années
    while population > seuil:
        an += 3
        population *= (1. - perte)
        print("{0} {1}".format(an, population))

    print("Au bout de {0} ans, la population est inférieure à 0.1% de la population initiale.".forma

if __name__ == "__main__":
    prog_population()
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" minmax  -  11.03.2014 """

from random import random

def minimax(maListe):
    """ Recherche du maximum et du minimum dans une liste pseudo-alétoire """

    # recherche du maximum et du minimum
    # initialisation
    mini = maListe[0]
    maxi = maListe[0]

    for i in range(n):
        # recherche du minimum
        if maListe[i] < mini:
            mini = maListe[i]

        # recherche du maximum
        if maListe[i] > maxi:
            maxi = maListe[i]

    print("maximum = {0}".format(maxi))
    print("minimum = {0}".format(mini))

    # avec les fonctions min() et max()
    print("maximum = {0}".format(max(maListe)))
    print("minimum = {0}".format(min(maListe)))

if __name__ == "__main__":
    # nombre de points
    n = int(raw_input("entrer n : "))
    print("n = {0}".format(n))

    # remplissage d'une liste pseudo aleatoire de nombres entre 0 et 100
    maListe = [100. * random() for i in range(n)]
    # ou avec des entiers
    # maListe = [randint(0, 100) for i in range(n)]

    minimax(maListe)
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" marche_aleatoire  -  11.03.2014 """

from random import random

def marche_aleatoire():
    """ Marche aléatoire dans un plan 2D """

    # nombres de pas
    npas = 1000
    amplitude = .1

    # liste pour les coordonnées
    x = list()
    y = list()

    # initialisation
    x.append(0.)
    y.append(0.)

    # marche aléatoire
    for i in range(npas):
        wx = 2. * random() - 1.
        wy = 2. * random() - 1.
        x.append(x[i] + amplitude * wx)
        y.append(y[i] + amplitude * wy)

    # représentation avec matplotlib
    import matplotlib.pyplot as plt

    plt.plot(x, y, "r-", label = "trajectoire")
    plt.title("Marche aleatoire dans un plan 2D")
    plt.axes().set_aspect("equal")
    plt.grid()
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.show()

if __name__ == "__main__":
    marche_aleatoire()
```

Code Python 4.25: Calcul du nombre π par la méthode Monte Carlo.

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" pi  -  14.03.2014

    Calcul du nombre pi par Monte Carlo.
    Principe : La probabilité qu'un point de coordonnées (x,y)
    avec x et y dans l'intervalle [0;1] soit dans le quart de
    cercle de centre (0,0) et de rayon 1 est égal au rapport des
    aires du quart de cercle de rayon 1 et du carré de largeur 1
    soit pi/4. """

import math
from random import random

def prog_pi(ntirage = 100):
    """ calcul du nombre pi """

    # initialisation
    n = 0

    # calcul monte carlo
    for i in range(ntirage):
        x = random()
        y = random()

        if x**2 + y**2 < 1.0:
            n += 1

    # affichage des résultats
    piCalcule = 4.0 * float(n) / float(ntirage)
    print("pi          = {0}".format(piCalcule))
    print("% d'erreur = {0}".format((math.pi - piCalcule) / math.pi * 100))

if __name__ == "__main__":
    # lecture du nombre de tirage
    ntirage = int(raw_input("entrer le nombre de tirage : "))
    print("ntirage = {0}".format(ntirage))

    prog_pi(ntirage)
```

Code Python 4.26: Intégration par la méthode des trapèzes.

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" trapeze  -  11.03.2014

Intégration par la méthode des trapèzes. L'intégration analytique
de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
application numérique -2.525369

voir également : scipy.integrate.trapz
"""

from math import exp

def trapeze(a, b, npas):
    """ integration par la methode des trapeze """

    # initialisation
    integrale = 0.
    pas = (b - a) / float(npas)

    # calcul de l'integrale
    for i in range(npas):
        x = a + float(i) * pas
        integrale += pas * (f(x) + f(x + pas)) / 2.

    return integrale

def f(x):
    """ fonction numerique utilisée """
    return (x**2 - 3. * x - 6.) * exp(-x)

if __name__ == "__main__":
    # lecture de l'intervalle
    a = float(raw_input("entrer a : "))
    print("a = {0}".format(a))

    b = float(raw_input("entrer b : "))
    print("b = {0}".format(b))

    # nombre de segments
    npas = int(raw_input("entrer le nombre de pas : "))
    print("npas = {0}".format(npas))

    integrale = trapeze(a, b, npas)

    analytique = -2.525369
    print("Résultat  = {0}".format(integrale))
    print("Résidu    = {0}".format(analytique - integrale))
    print("Précision = {0}".format((analytique - integrale) / analytique * 100))
```

Code Python 4.27: Intégration par la méthode de Simpson.

```
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" simpson  -  11.03.2014

Intégration par la méthode de Simpson. L'intégration analytique
de la fonction (x^2 - 3x - 6) e^-x entre 1 et 3 donne comme
application numérique -2.525369

voir également : scipy.integrate.simps
"""

from math import exp

def simpson(a, b, npas = 20):
    """ integration de simpson """

    # initialisation
    integrale = 0.0
    pas = (b - a) / float(npas)
    x = a

    # calcul de l'integrale
    while x < b:
        integrale += pas / 3. * (f(x) + 4. * f(x + pas) + f(x + 2. * pas))
        x += 2. * pas

    return integrale

def f(x):
    """ fonction numérique utilisée """
    return (x**2 - 3 * x - 6) * exp(-x)

if __name__ == "__main__":
    # lecture de l'intervalle
    a = float(raw_input("entrer a : "))
    print("a = {0}".format(a))

    b = float(raw_input("entrer b : "))
    print("b = {0}".format(b))

    # nombre de pas
    npas = int(raw_input("entrer le nombre de pas : "))
    print("npas = {0}".format(npas))

    integrale = simpson(a, b, npas)

    analytique = -2.525369
    print("Résultat  = {0}".format(integrale))
    print("Résidu    = {0}".format(analytique - integrale))
    print("Précision = {0}".format((analytique - integrale) / analytique * 100))
```

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

```python
#!/usr/bin/env python
# -*- coding=utf-8 -*-

""" schmidt  -  13.03.2014 """

from math import sqrt

def schmidt():
    """ Procédé d'orthogonalisation de Gram-Schimdt.
        Soit u et v deux vecteurs. On cherche le vecteur vp le plus
        proche de v orthogonal à u. """

    # vecteur u
    u = [1., 0., 0.]
    normu = sqrt(sum([ui**2 for ui in u]))

    # vecteur v
    v = [1., 2., 3.]

    # calcul du produit scalaire
    scalaire = sum([ui * vi for ui, vi in zip(u, v)])
    print("u.v = {0}".format(scalaire))

    # orthogonalisation de schmidt
    if scalaire != 0.0:
        vp = [vi - scalaire / normu * ui for ui, vi in zip(u, v)]

        # verification
        print("u.vp = {0}".format(sum([ui * vpi for ui, vpi in zip(u, vp)])))
        print("vp = {0}".format(vp))

if __name__ == "__main__":
    schmidt()
```

Algorithmes AlgoBox

Codes en FORTRAN

Codes en Python