Sous-programmes (suite)

Sous-programmes

Activité 1

Soit le programme suivant :

Algorithme

```
Algorithme Prog
Début
  Ecrire("Donner x ? "); Lire(x)
  y ← 3 * x * x - 5 * x + 7 // étape 2
  Ecrire(y)
Fin
```

a. Transformer l'étape 2 de l'algorithme précédent en une fonction.
b. Transformer l'étape 2 de l'algorithmeen une procédure.
c. Ecrire l'algorithme d'un programme qui affiche le tableau de valeurs de cette fonction dans un intervalle [a, b]. Le tableau contient n valeurs, 1 ≤ n ≤ 20.
d. Implémenter l'algorithme.

Ancien Régime

Question A

Transformer l'étape 2 de l'algorithme précédent en une fonction.

Algorithme

```
// Fonction
def fn f(a : réel):réel
  f ← 3 * a * a - 5 * a + 7
Fin

// Programme Principal
Début PP
  Ecrire("Donner x ? "); Lire(x)
  y ← f(x)
  Ecrire(y)
Fin
```

Explication

- La fonction accepte un paramètre de type réel et retourne un résultat de type réel.
- `a` est un **paramètre formel** qui est défini uniquement à l'intérieur de la fonction.
- `x` est un **paramètre effectif** qui indique la valeur passée réellement à la fonction. `x` est la valeur entrée par l'utilisateur et `y` est l'image de `x`.
- Lors de l'appel `f(x)` on copie la valeur de `x` dans la variable `a` de la fonction. On calcule l'image de `a` et on retourne le résultat dans le **nom de la fonction**.

Question B

Transformer l'étape 2 de l'algorithme en procédure.

Algorithme

```
// Procédure
def proc f(a : réel; var b : réel)
  b ← 3 * a * a - 5 * a + 7
Fin

// Programme Principal
Début PP
  Ecrire("Donner x ? "); Lire(x)
  f(x, y)
  Ecrire(y)
Fin
```

Explication

- La procédure accepte deux paramètres, le premier paramètre représente le réel dont on veut calculer l'image, le deuxième paramètre représente le résultat de la fonction réelle f.
- `a` et `b` sont des **paramètres formels** qui sont définis uniquement à l'intérieur de la procédure.
- `a` est un paramètre passé par valeur, c-à-d qu'il s'agit d'une copie de `x`.
- `a` est passé par valeur car la procédure ne vas pas opérer aucun changement sur sa valeur.
- `b` est un paramètre passé par variable, c-à-d, les changements opérés sur la variables `b` dans la procédure sont propagés à la variable `y` du programme principal.
- `x` et `y` sont des **paramètres effectifs** qui indiquent les valeurs réelles passées à la procédure. `x` est la valeur entrée par l'utilisateur et `y` est l'image de `x`.
- Lors de l'appel `f(x, y)` on copie la valeur de `x` dans la variable `a` et on indique que `b` est le surnom de `y`, c-à-d `b` et `y` sont deux noms d'un même objet. On calcule l'image de `a` et on retourne le résultat dans le **la variable b qui n'est autre que** `y`.

Question C

Ecrire l'algorithme d'un programme qui affiche le tableau de valeurs de cette fonction dans un intervalle [a, b]. Le tableau contient n valeurs, 2 ≤ n ≤ 20.

Algorithme

```
Début PP
  SaisirIntervalle(a, b)
  Saisir(n)
  RemplirTableaux(a, b, n, tx, ty)
  AfficherPoints(n, tx, ty)
Fin

Def Proc SaisirIntervalle(var a, b : réel)
  Ecrire("Donner a ? "); Lire(a)
  Répéter
    Ecrire("Donner b > ", a, " ? "); Lire(b)
  Jusqu'à a < b
Fin

Def Proc Saisir(var n : entier)
  Répéter
    Ecrire("Donner n > 1 ? "); Lire(n)
  Jusqu'à (2 ≤ n ≤ 20)
Fin
```

PP

T.D.N.T

TDNT
tab = tableau de 20 réel

T.D.O.G

Objet	Type/Nature
a, b n tx, ty	réel entier tab
SaisirIntervalle Saisir RemplirTableaux AfficherPoints	Procédure

RemplirTableaux

```
Def proc RemplirTableaux(a, b: réel; n: entier; var tx, ty: tab)
  pas ← (b - a) / (n - 1)
  Pour i de 0 à n-1 Faire
    tx[i+1] ← a + pas * i
    ty[i+1] ← f(tx[i+1])
  Fin Pour
Fin

Def Proc AfficherPoints(n : entier; tx, ty : tab)
  Pour i de 1 à n Faire
    Ecrire("f(",tx[i],") = ", ty[i])
  Fin Pour
Fin
```

T.D.O.L

Objet	Type/Nature
pas i	réel entier
f	Fonction

AfficherPoints

T.D.O.L

Objet	Type/Nature
i	entier

Nouveau Régime

Question A

Transformer l'étape 2 de l'algorithme précédent en une fonction.

Algorithme

```
// Fonction
Fonction f(a : réel):réel
  Retourner 3 * a * a - 5 * a + 7
Fin

// Programme Principal
Début PP
  Ecrire("Donner x ? "); Lire(x)
  y ← f(x)
  Ecrire(y)
Fin
```

Explication

- La fonction accepte un paramètre de type réel et retourne un résultat de type réel.
- `a` est un **paramètre formel** qui est défini uniquement à l'intérieur de la fonction.
- `x` est un **paramètre effectif** qui indique la valeur passée réellement à la fonction. `x` est la valeur entrée par l'utilisateur et `y` est l'image de `x`.
- Lors de l'appel `f(x)` on copie la valeur de `x` dans la variable `a` de la fonction. On calcule l'image de `a` et on retourne le résultat à l'aide de **Retourner**.

Question B

Transformer l'étape 2 de l'algorithme en procédure.

Algorithme

```
// Procédure
procédure f(a : réel; @b : réel)
  b ← 3 * a * a - 5 * a + 7
Fin

// Programme Principal
Début PP
  Ecrire("Donner x ? "); Lire(x)
  f(x, y)
  Ecrire(y)
Fin
```

Explication

- La procédure accepte deux paramètres, le premier paramètre représente le réel dont on veut calculer l'image, le deuxième paramètre représente le résultat de la fonction réelle f.
- `a` et `b` sont des **paramètres formels** qui sont définis uniquement à l'intérieur de la procédure.
- `a` est un paramètre passé par valeur, c-à-d qu'il s'agit d'une copie de `x`.
- `a` est passé par valeur car la procédure ne vas pas opérer aucun changement sur sa valeur.
- `b` est un paramètre passé par variable, c-à-d, les changements opérés sur la variables `b` dans la procédure sont propagés à la variable `y` du programme principal.
- `x` et `y` sont des **paramètres effectifs** qui indiquent les valeurs réelles passées à la procédure. `x` est la valeur entrée par l'utilisateur et `y` est l'image de `x`.
- Lors de l'appel `f(x, y)` on copie la valeur de `x` dans la variable `a` et on indique que `b` est le surnom de `y`, c-à-d `b` et `y` sont deux noms d'un même objet. On calcule l'image de `a` et on retrourne le résultat dans le **la variable b qui n'est autre que** `y`.

Question C

Ecrire l'algorithme d'un programme qui affiche le tableau de valeurs de cette fonction dans un intervalle [a, b]. Le tableau contient n valeurs, 2 ≤ n ≤ 20.

Algorithme

```
// Programme Principal
Algorithme tableau_valeurs
Début
  SaisirIntervalle(a, b)
  Saisir(n)
  RemplirTableaux(a, b, n, tx, ty)
  AfficherPoints(n, tx, ty)
Fin

Procédure SaisirIntervalle(@a, @b : réel)
  Ecrire("Donner a ? "); Lire(a)
  Répéter
    Ecrire("Donner b > ", a, " ? "); Lire(b)
  Jusqu'à a < b
Fin

Procédure Saisir(@n : entier)
  Répéter
    Ecrire("Donner n > 1 ? "); Lire(n)
  Jusqu'à (2 ≤ n ≤ 20)
Fin

Procédure RemplirTableaux(a, b: réel; n: entier; @tx, @ty: tab)
  pas ← (b - a) / (n - 1)
  Pour i de 0 à n-1 Faire
    tx[i] ← a + pas * i
    ty[i] ← f(tx[i])
  Fin Pour
Fin

Procédure AfficherPoints(n : entier; tx, ty : tab)
  Pour i de 0 à n-1 Faire
    Ecrire("f(",tx[i],") = ", ty[i])
  Fin Pour
Fin
```

PP

T.D.N.T

TDNT
tab = tableau de 20 réel

T.D.O.G

Objet	Type/Nature
a, b n tx, ty	réel entier tab
SaisirIntervalle Saisir RemplirTableaux AfficherPoints	Procédure

RemplirTableaux

T.D.O.L

Objet	Type/Nature
pas i	réel entier
f	Fonction

AfficherPoints

T.D.O.L

Objet	Type/Nature
i	entier