

Glossaire Axiomatique Impérative

Karine Zampieri, Stéphane Rivière

Unisciel  algoprogram  Version 22 mai 2018

Table des matières

| | | |
|----------|--|-----------|
| 1 | A | 3 |
| 1.1 | Accès indiciel | 3 |
| 1.2 | Affectation interne | 3 |
| 1.3 | Affichage de résultats | 4 |
| 1.4 | Appel d'une fonction | 4 |
| 1.5 | Appel d'une procédure | 5 |
| 2 | C | 5 |
| 2.1 | Commentaire | 5 |
| 3 | D | 5 |
| 3.1 | Déclaration et initialisation d'un tableau | 5 |
| 3.2 | Déclaration et initialisation d'un k-tableau | 6 |
| 3.3 | Déclaration de variables | 7 |
| 3.4 | Définition de constante | 7 |
| 3.5 | Définition d'une énumération | 7 |
| 3.6 | Définition d'un type structuré | 8 |
| 3.7 | Détection de fin de contenu | 8 |
| 4 | E | 9 |
| 4.1 | Écriture sur un canal de sortie | 9 |
| 5 | F | 9 |
| 5.1 | Fermeture d'un canal d'entrées/sorties | 9 |
| 5.2 | Fonctions caractère | 9 |
| 5.3 | Opérations de chaînes | 10 |
| 5.4 | Fonctions mathématiques | 11 |
| 5.5 | Formats d'édition | 12 |
| 6 | I | 12 |
| 6.1 | Inclure un fichier | 12 |
| 6.2 | Instruction composée | 13 |
| 6.3 | Instruction de retour | 14 |

| | | |
|-----------|--|-----------|
| 7 | L | 14 |
| 7.1 | Lecture depuis un canal d'entrée | 14 |
| 8 | O | 15 |
| 8.1 | Opérateurs arithmétiques | 15 |
| 8.2 | Opérateurs de comparaison | 15 |
| 8.3 | Opérateurs logiques | 16 |
| 8.4 | Opérateur Si-expression | 17 |
| 8.5 | Priorité des opérateurs | 17 |
| 8.6 | Ouverture d'un canal d'entrée/sortie | 17 |
| 9 | P | 18 |
| 9.1 | Paramètres formels | 18 |
| 9.2 | Primitives | 19 |
| 9.3 | Profil de fonction | 19 |
| 9.4 | Profil de procédure | 20 |
| 10 | R | 20 |
| 10.1 | Répétitive Itérer | 20 |
| 10.2 | Répétitive Pour | 21 |
| 10.3 | Répétitive Répéter | 22 |
| 10.4 | Répétitive TantQue | 22 |
| 10.5 | Ruptures de séquence (de bloc) | 23 |
| 11 | S | 24 |
| 11.1 | Saisie de données | 24 |
| 11.2 | Schéma d'une fonction | 24 |
| 11.3 | Schéma d'une procédure | 25 |
| 11.4 | Sélective Selon (listes de valeurs) | 25 |
| 11.5 | Sélective Si | 26 |
| 11.6 | Sélective Si-Alors | 27 |
| 11.7 | Sélective Si-Sinon-Si | 27 |
| 11.8 | Structure générale | 28 |
| 11.9 | Synonyme de type | 28 |
| 12 | T | 29 |
| 12.1 | Types intégrés | 29 |

Python - Glossaire Axiomatique Impérative

1 A

1.1 Accès indicial



Accès indicial

```
tab[k]
```

Explication

Accède à la case d'indice k d'un tableau tab .

Le temps d'accès à l'élément est fixe.

Numérotation des cases

Chaque langage de programmation possède sa propre convention.

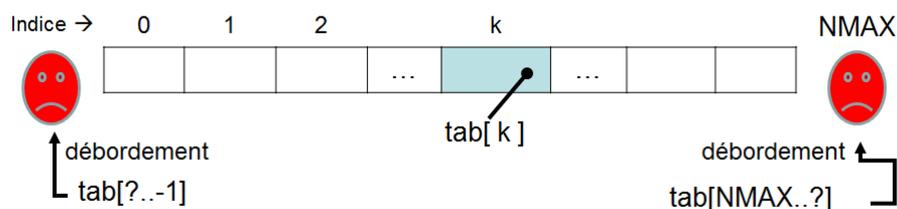
- **alg** : Les cases sont numérotées de 1 (par défaut) à $TMAX$ (taille du tableau).
- **C/C++, Java, Python** : Ils commencent à indiquer un tableau à partir de 0. Ce principe est dit **l'indexation en base 0**.
- **Basic** : Il débute la numérotation à partir de 1 ou 0.
- **Ada** : Il permet de numérotter les cases à partir d'une valeur quelconque.



Dépassement des bornes

Les langages contrôlent le débordement des bornes d'un tableau et déclenchent une erreur qui généralement arrête le programme.

```
const int NMAX = ...
T tab [ NMAX ]
```



1.2 Affectation interne



Affectation interne

Opération qui **fixe une valeur** à une variable.



Affectation interne

```
nomVar = expression
```

Explication

Place la valeur de l'**expression** dans la zone mémoire de la variable de nom **nomVar**. En algorithmique, le symbole **<-** (qui se lit « devient ») indique le sens du mouvement : de l'expression située à droite **vers** la variable à gauche.

**Rappel**

Toutes les variables apparaissant dans l'**expression** doivent avoir été affectés préalablement. Le contraire provoquerait un arrêt de l'algorithme.

**Conversions implicites**

Il est de règle que le résultat de l'expression à droite du signe d'affectation soit de même type que la variable à sa gauche.

1.3 Affichage de résultats

**Affichage de résultats**

```
print(expr1,...,exprN,end="") # SANS retour de ligne
print(expr1,expr2,...,exprN) # AVEC retour de ligne
```

Explication

Ordonne à la machine d'afficher les **valeurs** des expressions **exprI**. Par défaut, elles sont séparées par un espace et se terminera par un saut à la ligne. Le paramètre **sep** permet de préciser le séparateur.

**Python : print**

Avec PYTHON 3x, **print** devient une fonction : il faut donc mettre des parenthèses autour de ce que l'on souhaite afficher contrairement à PYTHON 2x.

**Variables devant être initialisées**

On ne peut *afficher* que des expressions dont les variables qui la composent ont été affectées préalablement.

1.4 Appel d'une fonction

**Appel d'une fonction**

```
v = nomFcn( a1, ..., aN )
```

Explication

Appelle (on dit aussi *invoque*) la fonction **nomFcn** avec les (éventuels) arguments **aI**. La valeur retournée peut être utilisée en tant que macro-expression.

**Fonction = macro-expression**

Une fonction retourne **toujours** une information à l'algorithme appelant. C'est pourquoi l'appel d'une fonction **ne se fait jamais** à gauche du signe d'affectation.

1.5 Appel d'une procédure

**Appel d'une procédure**

```
r1, ..., m1, ... = nomSsp(d1, ..., m1, ...)
```

Explication

Appelle la procédure `nomSsp` avec les (éventuels) arguments figurant entre les parenthèses.

**Procédure = macro-instruction**

Une procédure étant une macro-instruction, un appel de procédure se fait obligatoirement **en dehors de toute expression de calcul**.

2 C

2.1 Commentaire

**Commentaire (narratif)**

Texte qui n'est **ni lu, ni exécuté** par la machine. Il est essentiel pour rendre plus lisible et surtout plus compréhensible un script par un être humain.

**Commentaire orienté ligne**

```
... # rend le reste de la ligne non-exécutable
```

**Commentaire orienté bloc**

```
"""
rend le code entouré non exécutable...
"""
```

3 D

3.1 Déclaration et initialisation d'un tableau

**Déclaration/Création d'un tableau**

```
nomTab = [valInitTypeElement for x in range(taille)]
```

Explication

Déclare une variable dimensionnée. Avec : `TypeElement` le type (simple ou non) des éléments constitutifs du tableau, `nomTab` l'identifiant et `taille` son nombre d'éléments. La taille doit être une valeur entière positive (littéraux ou expressions constantes).

**Déclaration et initialisation**

```
nomTab = [val1, ..., valN]
```

Explication

Déclare et initialise un tableau : la longueur de la liste détermine le nombre d'éléments.

**Python : Tableau numpy**

Pour déclarer/créer un tableau numpy

```
import numpy as np
nomTab = np.zeros(taille, TypeElement)
```

Pour déclarer/initialiser un tableau numpy :

```
import numpy as np
nomTab = np.array([val1, ..., valN], dtype=TypeElement, order='C')
```

3.2 Déclaration et initialisation d'un k-tableau

**Déclaration/Initialisation d'un k-tableau**

```
nomTab = [...[valInit for jK in range(tailleK)]\
           ... for j1 in range(taille1)]
```

Explication

Déclare un tableau k -dimensionnel de nom `nomTab` d'éléments de type `T`. Les `tailleI` sont des valeurs entières positives (littérales ou expressions constantes).

**Cas particulier d'un tableau bidimensionnel**

```
nomTab = [[valInit for j22 in range(taille2)]\
           for j1 in range(taille1)]
```

**Déclaration et initialisation**

```
nomTab = [[[v111, ..., v11k]], ...]]]
```

Explication

Déclare et initialise un tableau k -dimensionnel : la dimension de la composante est la longueur de la liste.

3.3 Déclaration de variables



Déclaration de variables

Consiste à associer un type de données à une ou un groupe de variables. Toute variable doit impérativement avoir été déclarée avant de pouvoir figurer dans une instruction exécutable.



Déclaration de variables

```
nomVar = expression  
nomVar1, nomVar2, ... = expr1, expr2, ...
```

Explication

Déclare des variables d'identifiants `nomVarI` (le nom) de type `TypeVar`. Le type `TypeVar` est défini implicitement (*typage dynamique*) par la valeur de l'`expression`. La fonction intégrée `type(nomVar)` permet de connaître son type.

3.4 Définition de constante



Constante

Littéral à lequel est associé un **identifiant** (par convention, écrit en MAJUSCULES) afin de rendre plus lisible et simplifier la maintenance d'un script. C'est donc une information pour laquelle nom, type et valeur sont figés.



Définition de constante

```
nomConst = expression
```

Explication

Définit la constante d'identifiant `nomConst` de type `TypeConst` et lui affecte une valeur (littéral ou expression) spécifiée. Le type `TypeConst` est défini implicitement par la valeur de l'expression.



Python

Les constantes n'existent pas en PYTHON : nous utiliserons donc des **variables** globales.

3.5 Définition d'une énumération



Type énuméré

Définit un ensemble de constantes entières associées une à une à des identifiants ou *énumérateurs*. Deux avantages :

- Une indication claire des possibilités de la variable lors de la déclaration.
- Une lisibilité du code grâce à l'utilisation des valeurs explicites.



Définition d'une énumération

Explication

Introduit `NomType` dont les valeurs discrètes sont `nomVal1`, `nomVal2`, etc.

Quid des langages de programmation ?

Chaque langage de programmation propose sa propre technique de conversion de valeurs.

- Certains langages (comme JAVA) proposent un type énuméré complet.
- D'autres (comme C et C++) proposent un type énuméré incomplet mais qui permet néanmoins une écriture comme celle ci-dessus.
- D'autres langages ne proposent rien. Pour ces derniers, l'astuce est de définir des constantes entières qui vont permettre une écriture proche de celle ci-dessus (mais sans une déclaration explicite).

3.6 Définition d'un type structuré



Structure, Champ

Une **structure** permet de regrouper une ou plusieurs variables, de n'importe quel type (structure de données **hétérogène**), dans une entité unique et de la manipuler comme un tout. Chaque élément, appelé **champ** de la structure, possède un nom unique.



Définition d'un type structuré

```
class TypeStruct:
    def __init__(self):
        self.nomChamp1 = valeurType1
        self.nomChamp2 = valeurType2
        ...
```

Explication

Crée un nouveau type nommé `TypeStruct` à partir d'autres types élémentaires ou composés déjà définis `TypeI` et d'identifiants respectifs `nomChampI`.

3.7 Détection de fin de contenu



Détection de fin de contenu

Il n'y a pas de fonction spécifique.

Tant que le flux `f` est valide, **ce n'est pas** la fin de fichier.



Attention

La primitive n'est applicable qu'aux canaux associés en mode `Lecture`. Toute invocation de la primitive sur un canal associé à un document ouvert en mode `Ecriture` ou `Ajout` cause l'arrêt d'exécution de l'algorithme.

4 E

4.1 Écriture sur un canal de sortie



Écriture sur un canal de sortie

```
f.writeLine(expr)
```

Explication

Rajoute des données dans le fichier référencé par la variable `f`, ouvert en écriture, les valeurs des expressions `expr`.



Remarque

Le canal d'entrées/sorties doit obligatoirement être associé à un document ouvert en mode `Ecriture` ou `Ajour`. Toute tentative d'écriture visant un canal d'entrées/sorties associé à un document ouvert en mode `Lecture` provoque l'arrêt d'exécution de l'algorithme.

5 F

5.1 Fermeture d'un canal d'entrées/sorties



Fermeture d'un canal

```
f.close()
```

Explication

Ferme le canal d'entrées/sorties `f` précédemment ouvert et purge toutes les mémoires tampon. Dans le cas où le fichier a été ouvert en écriture, cette primitive place la marque spéciale de fin de fichier dans l'élément courant. Une fois le fichier fermé, il n'est plus permis de l'utiliser.



Remarque

Un fichier créé et non refermé risque de contenir des données aléatoires et invalides.

5.2 Fonctions caractère



Fonctions caractère

```
chr(n) # caractère dont le code ASCII est l'entier n compris entre 32 et 255  
ord(c) # code ASCII entre 32 et 255 du caractère c
```

5.3 Opérations de chaînes



Longueur d'une chaîne

Nombre de caractères dans la chaîne.



Concaténation de deux chaînes

Consiste à prendre ces deux chaînes et à les coller bout-à-bout.



Sous-chaîne d'une chaîne

Suite consécutive de caractères de la chaîne : c'est une partie (un morceau) de cette chaîne.



Comparer deux chaînes

C'est déterminer laquelle des deux précède l'autre pour l'ordre alphabétique des dictionnaires (encore appelé **ordre lexicographique**) où la chaîne vide "" est avant toutes les autres et où il faut tenir compte des lettres minuscules et majuscules ainsi que des caractères spéciaux. La comparaison de deux chaînes s'effectue **caractère par caractère de gauche à droite** jusqu'à rencontrer la fin d'une des deux chaînes ou une différence. La chaîne de caractères qui **précède** l'autre est celle qui, la première, a un caractère qui **précède** le caractère correspondant de l'autre chaîne. En cas d'égalité permanente, la chaîne la plus courte **précède** la chaîne la plus longue.



Comparaison de chaînes

Les opérateurs == et != servent à comparer des chaînes. Les opérateurs <, >, <= et >= permettent de les classer dans l'ordre alphabétique.



Opérations de chaînes

```
chn1 + chn2 //concaténation de chaînes
```



Quelques fonctions

```
len(chn) #longueur de la chaîne
chn[p] #chaîne constituée du p-ème caractère de la chaîne
chn[n:m] #chaîne constituée des caractères n à m (exclus) de la chaîne
```



Remarque

L'« addition » de chaînes n'est pas une opération commutative car la chaîne $x + y$ n'est pas identique à la chaîne $y + x$.



Attention

Les positions dans les chaînes commencent à 0. Si l'indice est négatif, l'extraction s'effectue par rapport à la fin. Ainsi $x[-1]$ désigne le dernier caractère de la chaîne x .

| Instructions | Signification |
|-------------------------|--|
| <code>x.count(c)</code> | Nombre de fois que la lettre <code>c</code> apparaît dans la chaîne <code>x</code> |
| <code>x.index(c)</code> | Première position de la lettre <code>c</code> dans la chaîne <code>x</code> |
| <code>x.lower()</code> | Transforme tous les caractères de la chaîne <code>x</code> en minuscules |
| <code>x.upper()</code> | Transforme tous les caractères de la chaîne <code>x</code> en majuscules |



Autres instructions

Le tableau suivant présente des instructions qui permettent de transformer ces chaînes ou d'en extraire des informations comme, par exemple, compter les occurrences d'une lettre.

5.4 Fonctions mathématiques



Fonctions mathématiques

Elles agissent sur des paramètres à valeurs réelles et donnent un résultat réel.



Pour les utiliser

```
import math
```



Quelques Fonctions mathématiques

| Fonctions Mathématiques | Signification | Équivalent Python |
|-------------------------|---|-----------------------------|
| π | valeur du nombre π | <code>math.pi</code> |
| $\text{acos}(x)$ | Angle (exprimé en radians) dont le cosinus est égal à x | <code>math.acos(x)</code> |
| $\text{asin}(x)$ | Angle (exprimé en radians) dont le sinus est égal à x | <code>math.asin(x)</code> |
| $\text{atan}(x)$ | Angle (exprimé en radians) compris entre $-\pi/2$ et $\pi/2$ dont la tangente est égale à x | <code>math.atn(x)</code> |
| $\lceil x \rceil$ | Réel de l'entier supérieur | <code>math.ceil(x)</code> |
| $\cos(x)$ | Cosinus de x (exprimé en radians) | <code>math.cos(x)</code> |
| e^x | Exponentielle de x (base e) | <code>math.exp(x)</code> |
| $ x $ | Valeur Absolue de x | <code>math.fabs(x)</code> |
| $\lfloor x \rfloor$ | Réel de l'entier inférieur à x | <code>math.floor(x)</code> |
| $\log(x)$ | Logarithme naturel (base e) | <code>math.log(x)</code> |
| x^y | Puissance | <code>math.pow(x, y)</code> |
| $\sin(x)$ | Sinus de x (exprimé en radians) | <code>math.sin(x)</code> |
| \sqrt{x} | Racine carrée du nombre positif x | <code>math.sqrt(x)</code> |
| $\tan(x)$ | Tangente de x (exprimé en radians) | <code>math.tan(x)</code> |



Racine carrée

Attention de ne l'utiliser qu'avec un radicant positif.

5.5 Formats d'édition



Format d'édition d'une expression

Indique de quelle façon doit être cadrée l'expression à afficher. Il s'applique aux valeurs de type `Chaîne`, `Entier` ou `Réel`.



Formats d'édition

```
print("{largeur1}".format(exprChaîne))
print("{largeur1}".format(exprEntier))
print("{largeur1:.largeur2f}".format(exprReel))
```

Explication

Définit le format d'édition. L'entier `largeur1` indique sur combien de caractères doit être écrite l'expression. L'entier `largeur2` précise le nombre de chiffres après le point décimal des réels.

Règle d'affichage

Si le format est :

- **Égal** à la longueur nécessaire à l'édition de la valeur : la valeur est écrite telle quelle.
- **Inférieur** à la longueur : il est ignoré et la valeur est écrite sur la longueur nécessaire.
- **Supérieur** à la longueur : le système effectue un cadrage de la valeur à afficher à l'intérieur du format qui lui a été spécifié. Les données numériques sont cadrées à droite sur le point décimal, et les données alphanumériques cadrées à gauche.

6 I

6.1 Inclure un fichier



Python : Inclure un fichier

```
import NomFichier
```

Explication

Importe dans le script courant, les modules (procédures et fonctions) définis dans le fichier `NomFichier`. L'emploi d'un module se fera par :

```
Nomfichier.ssprg(liste_d_arguments)
```

6.2 Instruction composée



Instruction

Ordre donné à l'ordinateur qui a pour effet de changer l'état de la mémoire ou le déroulement du programme ou bien de communiquer avec les unités périphériques (clavier, écran, imprimante, etc.).



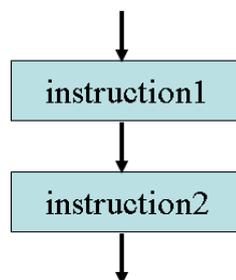
Instruction composée ou Bloc

Regroupement syntaxique de 0, 1 ou plusieurs instructions (et déclarations) comme une unique instruction.



Séquentialité

Les algorithmes et programmes présentés sont exclusivement séquentiels : l'*instruction2* ne sera traité qu'une fois l'exécution de l'*instruction1* achevée.



Bloc

```

Bloc: #<- deux points
  instruction1 #indentation
  instruction2 #même indentation
  ...
  
```



Conventions usuelles

Savoir présenter un script, c'est montrer que l'on a compris son exécution.

- Chaque ligne comporte une seule instruction.
Le point-virgule « ; » est le séparateur d'instructions.
- Un bloc est défini par des lignes de **même indentation** (espaces ou tabulations).



Python : L'indentation

C'est une erreur de syntaxe d'indenter deux instructions d'une valeur différente dans un même corps de bloc. De même, ne mélangez pas les tabulations et les espaces car cela donne lieu à des erreurs de compilation qui peuvent paraître incompréhensibles.



Python : Instruction

Une instruction se termine à la fin d'une ligne mais elle peut se poursuivre sur plusieurs lignes si ces dernières se terminent par un antislash (\), si une paire de (), [], {} ou une chaîne encadrée de guillemets triples n'est pas refermée. Plusieurs instructions simples peuvent apparaître sur une même ligne si elles sont séparées par un point-virgule (;).

6.3 Instruction de retour



Instruction de retour

```
return (expr1, expr2, ..., exprN)
```

Explication

Renvoie (retourne) au module appelant la valeur du n-uplet (`expr1`, `expr2`, ..., `exprN`) placée à la suite du mot `return`.



Dans une fonction

Il doit **toujours** y avoir l'exécution d'une primitive `return`, et ceci quelles que soient les situations (conditions).

En effet, si dans un cas particulier, la fonction s'exécute sans être passée par cette primitive, ceci révèle une incohérence dans la conception de votre fonction car celle-ci aura une valeur inconnue et aléatoire.

7 L

7.1 Lecture depuis un canal d'entrée



Lecture depuis un canal d'entrée

```
s = f.readline()
```

Explication

Lit une ligne de texte dans le fichier référencé par la variable `f`, ouvert en lecture.



Remarque

Le canal d'entrées/sorties doit obligatoirement être associé à un fichier ouvert en mode `Lecture`. Toute tentative de lecture visant un canal d'entrées/sorties associé à un fichier ouvert en mode `Ecriture` ou `Ajout` cause l'arrêt d'exécution de l'algorithme.

8 O

8.1 Opérateurs arithmétiques



Opérateurs arithmétiques

Dits aussi **opérateurs algébriques**, ils agissent sur des opérandes de type numérique.



Opérateurs arithmétiques

| Opérateur Mathématique | Signification | Équivalent Python |
|------------------------|---------------------------------------|-------------------|
| + | (unaire) valeur | +a |
| - | (unaire) opposé | -a |
| + | addition | a + b |
| - | soustraction | a - b |
| * | multiplication | a * b |
| / | division décimale | a / b |
| div | division entière | a // b |
| mod | modulo (reste de la division entière) | a % b |
| ^ | élévation à la puissance | a ** b |



Python : Division euclidienne

Sous PYTHON 2, la division / appliquée à des entiers retourne le résultat de la division entière. Sous PYTHON 3, l'opérateur / donne toujours le résultat de la division réelle et l'opérateur // donne le résultat de la division entière.



Division euclidienne, cas des négatifs

Il n'y a pas unicité du quotient et du reste lorsque le dividende ou le diviseur sont négatifs. Si a et b sont deux entiers relatifs dont l'un au moins est négatif, il y a plusieurs couples (q, r) tels que $a = b \times q + r$ avec $|r| < |b|$. Par exemple, si $a = -17$ et $b = 5$ alors $(q = -3, r = -2)$ ou $(q = -4, r = 3)$ sont deux solutions possibles. Habituellement, q et r sont choisis comme le quotient et le reste de la division entière de $|a|$ et $|b|$ affectés du signe approprié (celui permettant de vérifier $a = b \times q + r$ avec $|r| < |b|$). Dans l'exemple ci-avant, la solution retenue serait $(q = -3, r = -2)$. Par contre, la norme impose que la valeur de $(a \text{ div } b) * b + a \text{ mod } b$ soit égale à la valeur de a .



Division par zéro

Tout emploi de la division devra être accompagné d'une réflexion sur la valeur du dénominateur, une division par 0 entraînant toujours l'arrêt d'un script.

8.2 Opérateurs de comparaison



Opérateurs de comparaison

Dits aussi **opérateurs relationnels** ou **comparateurs**, ils agissent généralement sur des

variables numériques ou des chaînes et donnent un résultat booléen. Pour les caractères et chaînes, c'est l'ordre alphabétique qui détermine le résultat.

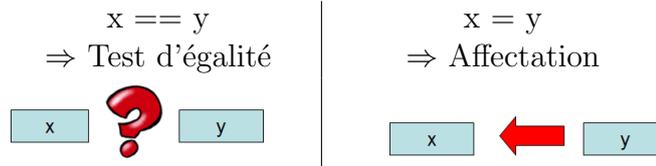


Opérateurs de comparaison

| Opérateur Mathématique | Signification | Équivalent | |
|---------------------------|-----------------------------|---------------|------------|
| | | Algorithmique | Python |
| < | (strictement) inférieur | $a < b$ | $a < b$ |
| \leq | inférieur ou égal | $a \leq b$ | $a \leq b$ |
| > | (strictement) supérieur | $a > b$ | $a > b$ |
| \geq | supérieur ou égal | $a \geq b$ | $a \geq b$ |
| = | égalité | $a = b$ | $a == b$ |
| \neq | différent de (ou inégalité) | $a \neq b$ | $a != b$ |



Distinguer == et =



A gauche Compare la valeur de x à celle de y et rend true si elles sont égales, false sinon (et donc **ne modifie pas** la valeur de x)

A droite Affecte à la variable x la valeur de y (et donc **modifie** la valeur de x)

8.3 Opérateurs logiques



Opérateurs logiques

Dits aussi **connecteurs logiques** ou **opérateurs booléens**, ils agissent sur des expressions booléennes (variables ou expressions à valeurs booléennes) pour donner un résultat du même type. Ils peuvent être enchaînés.



Opérateurs logiques

| Opérateur Mathématique | Signification | Équivalent | |
|---------------------------|-----------------------------------|---------------|--------------------|
| | | Algorithmique | Python |
| \neg | négation (unaire) | Non a | $\text{not } a$ |
| \wedge | conjonction logique | a Et b | $a \text{ and } b$ |
| \vee | disjonction logique (ou inclusif) | a Ou b | $a \text{ or } b$ |



Opérateur Ou-exclusif

Il n'y a pas d'opérateur OU-exclusif (xor) logique.

8.4 Opérateur Si-expression



Opérateur Si-expression

```
exprAlors if exprBool else exprSinon
```

Explication

Évalue l'expression logique `exprBool` et si elle est vérifiée, effectue l'expression `exprAlors`, sinon l'expression `exprSinon`. Les `exprAlors` et `exprSinon` doivent être du même type.



Remarque

Cette syntaxe très raccourcie doit être réservée à de petits tests.

8.5 Priorité des opérateurs



Python : Priorité des opérateurs

Les opérateurs de même priorité sont regroupés sur une même ligne.

| Priorité | Opérateur | | Signification | |
|----------------|---------------|------------|--|------------|
| | Algorithmique | Python | | |
| La plus élevée | - (unaire) | - (unaire) | Négation algébrique | |
| | ^ | ** | Puissance | |
| | * / div mod | * / // % | Multiplication, division, div. entière, modulo | |
| | + - | + - | Addition et soustraction | |
| | < <= > >= | < <= > >= | Opérateurs de comparaison | |
| | = <> | == != | Opérateurs d'égalité | |
| | Non | not | Négation logique | |
| | Et | and | Et logique | |
| | La plus basse | Ou | or | Ou logique |



Cas de combinaisons de Et et de Ou

Mettez des parenthèses :

```
(cond1 Et cond2) Ou cond3
est différent de
cond1 Et (cond2 Ou cond3)
```

En l'absence de parenthèses, le `Et` est prioritaire sur le `Ou`.

8.6 Ouverture d'un canal d'entrée/sortie



Ouverture d'un canal

```
import os
f = open(nomFich, utilisation)
```

Explication

Associe un canal d'entrées/sorties à un fichier en mode d'accès (*Utilisation*) : "r" (lecture), "w" (écriture) ou "a" (ajout). Le `nomFich` est une chaîne de caractères contenant le nom du fichier à ouvrir avec éventuellement le chemin d'accès à savoir le nom du disque et le chemin relatif ou absolu permettant d'atteindre le fichier. A défaut, le fichier doit être dans le dossier courant (habituellement le dossier où est sauvegardé le projet en exécution).

**Erreur à l'ouverture**

Le système peut être dans l'impossibilité d'ouvrir le fichier spécifié pour une ou l'autre des raisons suivantes :

- Tentative d'ouvrir un fichier inexistant en mode lecture.
- Tentative d'ouvrir un fichier qui est déjà ouvert.
- Tentative d'ouvrir un fichier sur un canal d'entrées/sorties invalide.
- Le nom du fichier est invalide : ceci peut être dû au dossier inexistant, au nom du fichier contenant des caractères interdits par le système d'exploitation ou à l'unité de stockage défectueuse ou non disponible.

Dans ce cas le système provoque l'interruption du programme et affiche un message d'erreur précisant la cause de l'erreur.

**Fichier en mode écriture**

L'ouverture efface automatiquement son contenu s'il existe déjà.

9 P

9.1 Paramètres formels

**Paramètres Entrants/Sortants/Mixtes**

Les paramètres **entrants** ou *données* :

- Ont une **valeur à l'entrée** du module.
- Et seront **consultés à l'intérieur** du module.

Les paramètres **sortants** ou *résultats* :

- Ont une **valeur indéterminée à l'entrée** du module.
- Et seront **utilisables après l'appel** du module.

Les paramètres **mixtes** ou *modifiés* :

- Ont une **valeur à l'entrée** du module.
- Et seront éventuellement **modifiés à l'intérieur** de celui-ci.

**Paramètres formels**

```
def nomSsp(d1, ..., m1, ...):
    ...
    return r1, ..., m1, ...
```

Explication

Les **d** sont des paramètres **donnés**, les **r** des **résultats** et **m** des **modifiés**.

PYTHON autorisant des résultats multiples (des n-uplets), les paramètres résultats et modifiés seront retournés par la fonction.

9.2 Primitives**Bibliothèque**

Ensemble de fonctionnalités ajoutées à un langage de programmation. Chaque bibliothèque décrit un thème.

**Pour les utiliser**

```
import nomBiblio # importe le module nomBiblio
from nomBiblio import desTrucsUtiles # importe uniquement ceux spécifiés
```

**Primitives**

Noms de fonctions (**abs**, **log**, **sin...**), d'opérateurs (**div**, **mod...**) ou de traitement (**afficher**, **saisir...**). Elles acceptent un ou plusieurs paramètres et jouent le même rôle syntaxique qu'un identifiant.

**Appel d'une primitive**

```
P(x,...) # procédure
r = F(x,...) # fonction
```

Explication

Appelle (on dit aussi **invoque**) la procédure **P** ou la fonction **F** avec les arguments **x...** Dans le cas de fonction, la valeur retournée peut être utilisée en tant que macro-expression.

9.3 Profil de fonction**Profil de fonction**

Constitué par le nom de la fonction, la liste des types des paramètres d'entrée et le type du résultat.

**Profil de fonction**

```
def nomFcn(param1, ..., paramN):
```

Explication

Annonce la fonction d'identifiant **nomFcn** ayant pour paramètres formels les **paramI**. La liste est vide si la fonction n'a pas besoin de paramètres.

**Remarque**

En théorie, le type de la valeur retournée peut être un type simple (entier, réel, booléen...), un type structuré, un tableau ou même un objet (ces types seront vus dans les modules suivants). En pratique il conviendra de s'en tenir aux limitations du langage utilisé.

**Remarque**

Les paramètres formels deviennent automatiquement des variables locales (cf. plus bas, @[Variable locale]) du module.

9.4 Profil de procédure

**Profil de procédure**

```
def nomSsp(parametres):
```

Explication

Définit le **profil** de la procédure de nom `nomSsp` ayant pour paramètres formels les `parametres` lesquels décrivent pour chaque paramètre, son nom, son type et sa caractéristique.

**Python : Valeur None**

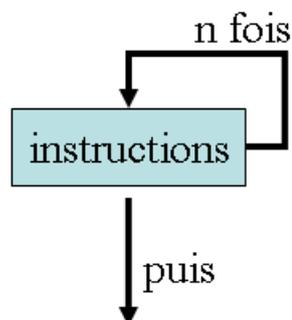
Une fonction retourne toujours une valeur, éventuellement la valeur `None`.

10 R

10.1 Répétitive Itérer

**Répétitive Itérer**

Elle traduit : Exécuter `n` fois les `instructions`, avec `n` un entier positif. Finitude assurée.





Répétitive Itérer

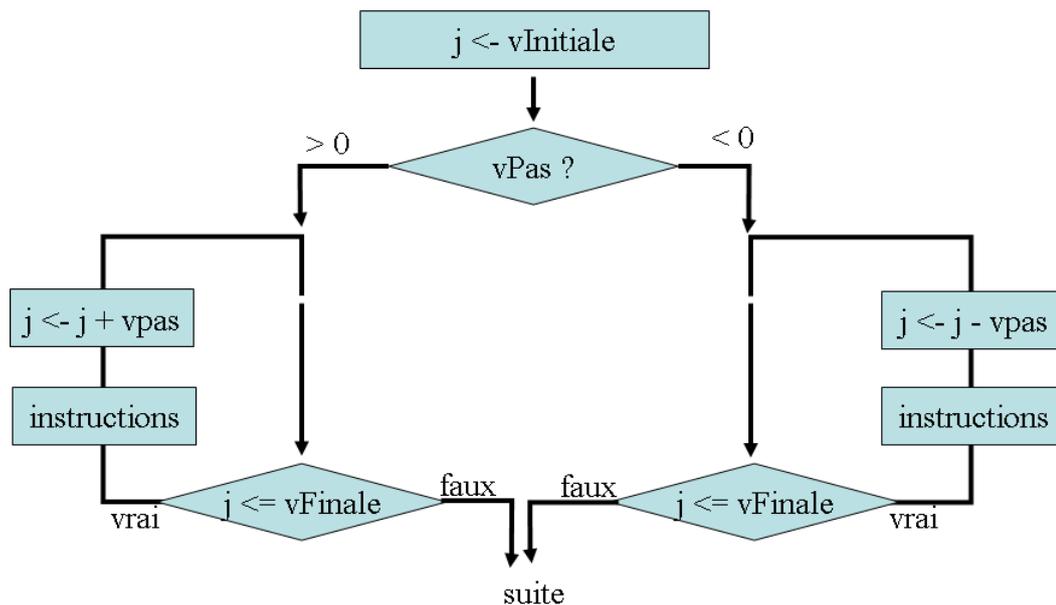
```
for j in range(1,n+1):
    instructions
```

10.2 Répétitive Pour



Répétitive Pour

Elle traduit : Exécuter les `instructions`, **Pour** une variable de boucle `nomVar` (entière ou réelle) dont le contenu varie de la valeur initiale `valDeb` à la valeur finale `valFin` par pas de `valPas` (par défaut de 1). Finitude assurée.



Terminologie

La variable utilisée dans la boucle **Pour** s'appelle la **variable de boucle**, **variable de contrôle**, **indice d'itération** ou **compteur de boucle**. En général, son nom se réduit simplement à une lettre, par exemple `j`.



Répétitive Pour

```
for nomVar in range([valDeb,] valFin+/-1 [, valPas]):
    instructions
```



Attention

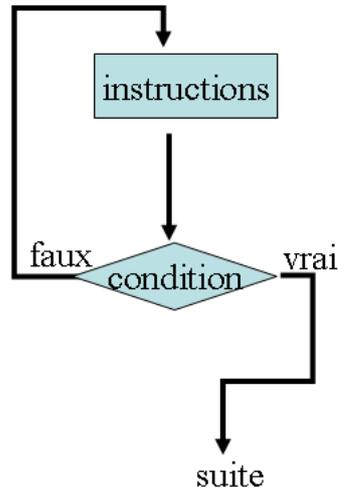
Par défaut l'entier `valDeb` vaut 0 et `valPas` vaut 1. L'entier `valFin` (deuxième paramètre) est exclu. Ainsi `range(1,11)` génère les entiers de 1 à 10.

10.3 Répétitive Répéter



Répétitive Répéter (répétition a-posteriori)

Elle traduit : Exécuter les **instructions** Jusqu'à ce que la **condition** est vraie.



Boucle infinie

La séquence **instructions** **doit** modifier la condition de telle manière qu'elle puisse devenir **vraie** pour arrêter l'itération.



Répétitive Répéter

```

while True:
    ...
    if not condition: break
    ...
  
```



Python

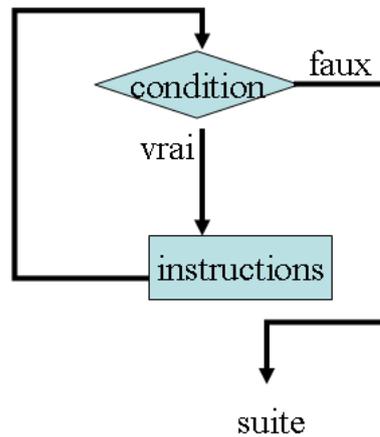
La répétitive **Répéter** n'existe pas en Python. Elle sera simulée par le code défini ci-avant en utilisant le **TantQue** et l'instruction de rupture **SortirSi**.

10.4 Répétitive TantQue



Répétitive TantQue (répétition a-priori)

Elle traduit : **TantQue** la **condition** est vraie, exécuter les **instructions**.
Si la **condition** est fausse dès le début, la tâche n'est jamais exécutée.



Boucle infinie

La séquence `instructions` **doit** modifier la condition de telle manière qu'elle puisse devenir **fausse**. Dans le cas contraire, la boucle va tourner sans fin (condition indéfiniment vraie) : c'est ce qu'on appelle une **boucle infinie**.



Répétitive TantQue

```
while condition:
    instructions
```

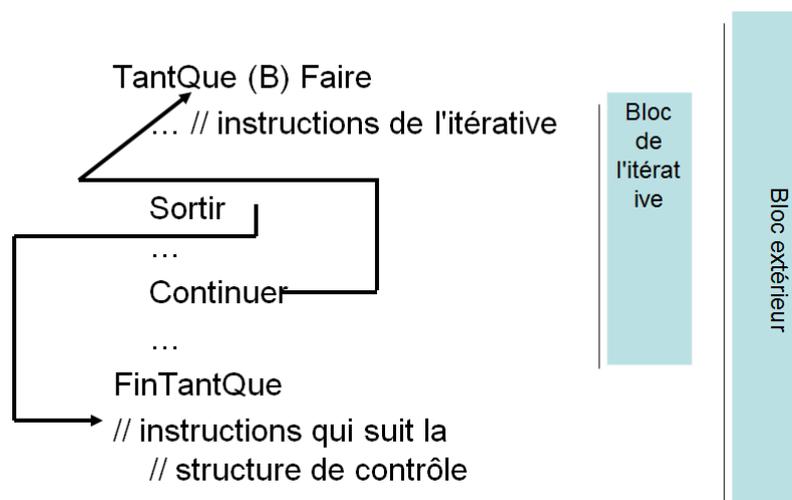
10.5 Ruptures de séquence (de bloc)



Ruptures de séquence (de bloc)

On en distingue deux :

- `Sortir` : Interrompt l'exécution de la structure de contrôle en provoquant un saut vers l'instruction qui suit la structure de contrôle.
- `Continuer` : Interrompt l'exécution des instructions du bloc d'une **répétitive** et provoque la ré-évaluation de la condition de continuation afin de déterminer si l'exécution de l'itérative doit être poursuivie (avec une nouvelle itération).



**Utilisez-les avec parcimonie**

Car elles ne permettent pas de réaliser une preuve formelle d'un algorithme (cf. @[Preuve et Notations asymptotiques]).

**Rupture Sortir**

```
break
```

**Rupture Continuer**

```
continue
```

11 S

11.1 Saisie de données

**Saisie de données**

```
nomVarI = input(["Invite"]) # lecture d'une chaîne
nomVarI = int(input(["Invite"])) # entrée typée d'un entier
nomVarI = float(input(["Invite"])) # entrée typée d'un réel
nomVarI = bool(input(["Invite"])) # entrée typée d'un booléen
nomVar1, ..., nomVarN = eval(input(["Invite"])) # typage dynamique
```

Explication

Ordonne à la machine de lire des valeurs `valI` depuis le clavier et de les stocker dans les variables `nomVarI` (qui doivent exister c.-à-d. déclarées).

**Remarque**

Par défaut, ce qui est tapé au clavier est envoyé à l'écran et temporairement placé dans un tampon pour permettre la correction d'erreurs de frappe. On peut donc se servir des touches [←] et [Suppr] pour effacer un caractère erroné ainsi que les flèches [<] et [>] pour se déplacer dans le texte.

11.2 Schéma d'une fonction

**Schéma d'une fonction**

```
def nomFcn(param1,param2,...):
    resultat = valeurInitiale
    calcul_du_resultat
    return resultat
```

Explication

Pour des questions de lisibilité et de preuve de programme, il vous est fortement recommandé d'adopter le schéma ci-dessus.

**Expression fonctionnelle**

```
def nomFcn(param1,param2,...):
    return expression
```

**Fonction lambda**

```
nomFcn = lambda parametresFormels : expression
```

Explication

Dans le cas d'une expression calculable directement, on peut regrouper le tout : on parle alors d'**expression fonctionnelle**.

11.3 Schéma d'une procédure

**Schéma d'une procédure**

```
def nomSsp(d1,...,m1,...):
    ...
    return r1,...,m1,...
```

Explication

Définit la procédure de nom `nomSsp`.

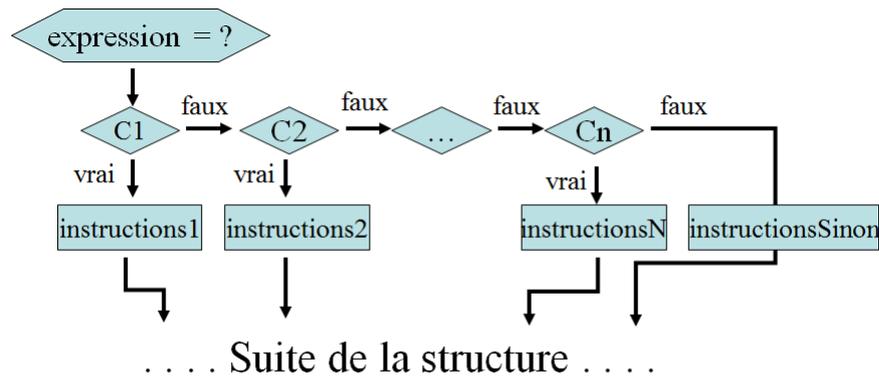
**Python**

Si aucune valeur n'est explicitement retournée, PYTHON retourne la valeur `None`.

11.4 Sélective Selon (listes de valeurs)

**Sélective Selon (listes de valeurs)**

Elle évalue l'`expression` et n'exécute que les `instructionsI` qui correspondent à la valeur ordinaire `ci` (c.-à-d. de type entier ou caractère). La clause `Cas Autre` est facultative et permet de traiter tous les cas non traités précédemment. Il s'agit de l'instruction multi-conditionnelle classique des langages.



Sélective Selon (listes de valeurs)

```

if expression in [C1, ...]:
    instructions1
elif expression in [C2, ...]:
    instructions2
elif ...
else:
    instructionsD
  
```



Remarque

Veillez à ne pas faire apparaître une même valeur dans plusieurs listes.



Python

Il n'existe pas de structure directe.

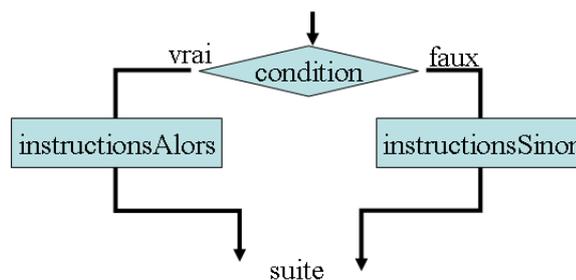
Il faut obligatoirement passer par des `if`, `elif`, `else`.

11.5 Sélective Si



Sélective Si

Elle traduit : **Si** la `condition` est vraie, exécuter les `instructionsAlors`, **Sinon** exécuter les `instructionsSinon`. Il s'agit d'un choix binaire : **une et une seule** des deux séquences est exécutée.



La `condition` peut être simple ou complexe (avec des parenthèses et/ou des opérateurs logiques `Et`, `Ou`, `Non`).



Sélective Si

```
if condition:
    instructionsAlors
else:
    instructionsSinon
```



Python

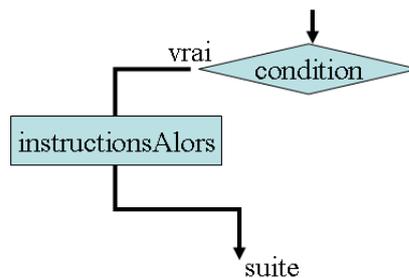
Notez l'absence du mot-clé `Alors`.

11.6 Sélective Si-Alors



Sélective Si-Alors

Forme restreinte de la structure `Si` (sans clause `Sinon`).



Sélective Si-Alors

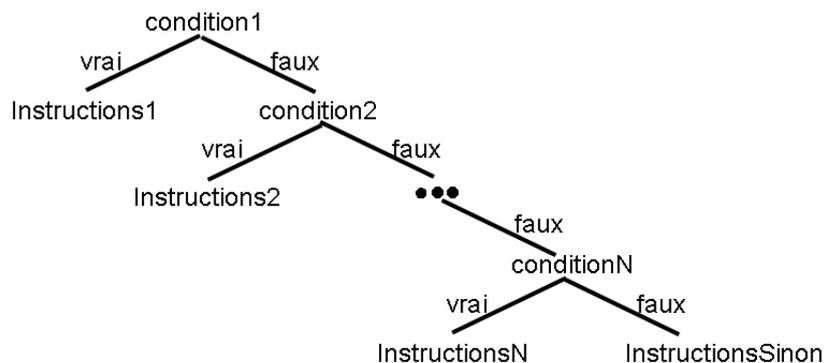
```
if condition:
    instructionsAlors
```

11.7 Sélective Si-Sinon-Si



Sélective Si-Sinon-Si

Elle évalue successivement la `conditionI` et exécute les `instructionsI` si elle est vérifiée. En cas d'échec des `n` conditions, exécute les `instructionsSinon`.





Sélective Si-Sinon-Si

```
if condition1:
    instructionsA1
elif condition2:
    instructionsA2
elif ...
...
elif conditionN:
    instructionsAn
else:
    instructionsSinon
```

11.8 Structure générale



Structure générale

```
from biblio import des_trucs_utiles
déclaration_des_objets_globaux
déclarations_et_définitions_de_fonctions_utiles

def PGPrincipal():
    corps_du_programme

PGPrincipal()
```

Explication

Un script est constitué par :

- Un **en-tête** qui demande d'inclure les modules indiqués.
- Un **corps** lequel contient une fonction (ici `PGPrincipal()`) définie comme étant la fonction « principale » .

Le script commence son exécution sur le bloc de la fonction `PGPrincipal` qui débute après les deux-points, se déroule séquentiellement et se termine sur la fin de l'indentation.



Conseil

On veillera à ce qu'un script tienne sur une vingtaine de lignes (donc, en pratique, sur un écran de 40 x 80 caractères ou une page). Ceci implique que, si votre script devait être plus long, il faudra le découper, comme nous le verrons plus loin.

11.9 Synonyme de type



Synonyme de type

Alias d'un type existant (lorsqu'un nom de type est trop long ou est difficile à manipuler).



Synonyme de type

N'existe pas.

**Typedef = Définition**

N'introduit pas de nouveau type mais un **nouveau nom** pour le type.

12 T

12.1 Types intégrés

**Types intégrés (*builtins*)**

Dits aussi **types de base**, **types fondamentaux** ou encore **types primitifs**, ils correspondent aux données qui peuvent être traitées directement par le langage.

**Types intégrés**

| Domaine | Algorithmique | Équivalent Python |
|--------------|---------------|--------------------|
| \mathbb{Z} | Entier | <code>int</code> |
| \mathbb{R} | Réel | <code>float</code> |
| \mathbb{B} | Booléen | <code>bool</code> |
| \mathbb{A} | Caractère | — |
| \mathbb{T} | Chaîne | <code>str</code> |

**Python : Le type Caractère**

Il n'existe pas : un caractère est simplement une chaîne de longueur 1.