Fiche mémoire algorithmique et Python

1. Instructions élémentaires

Les mots écrits en italique ou en gras sont à remplacer selon les besoins.

a. Variables

- **Saisie :** Demander à l'utilisateur la valeur d'une variable X :

  Saisir X

- **Affectation :** Donner une valeur à une variable X :

  X$\leftarrow$ *valeur*

  On peut lire cette instruction ainsi : « X prend la valeur *valeur* ».

b. Affichage

- Afficher la valeur d'une variable X :

  Afficher X

- Afficher une chaîne de caractère :

  Afficher "chaîne de caractère"

c. Instruction conditionnelle

- Sans utilisation d'un sinon :

  Si **condition** Alors
  | *Instructions si* **condition** *vérifiée*
  FinSi

- Avec utilisation d'un sinon :

  Si **condition** Alors
  | *Instructions si* **condition** *vérifiée*
  Sinon
  | *Instructions si* **condition** *non vérifiée*
  FinSi

d. Boucle non conditionnelle

Pour I allant de *valeur_initiale* à *valeur_finale* Faire
| *Instructions*
Fin Pour

La variable I sert de compteur. À la fin de chaque itération, la valeur de I est augmentée de 1.
Si I dépasse *valeur_finale*, on sort de la boucle.

e. Boucle conditionnelle

Tant que **condition** Faire
| *Instructions si* **condition** *est vérifiée*
Fin Tant que

Au début de chaque itération, la **condition** est testée. Si elle n'est pas vérifiée, on sort de la boucle.

2. Traduction en langage Python

Cette page donne la traduction en Python des diverses instructions données à la page précédente. Attention à l'**indentation** (espacement en début de ligne) dans les instructions `if`, `for` et `while`.

pseudo algorithmique	**Python**	**Remarques**

Saisir X	`X=input("X=?")`
	ou
	`X=int(input("X=?"))`
	ou
	`X=float(input("X=?"))`
		X est une chaîne de caractères
		
		X est un entier
		
		X est un flottant

X←Y	`X=Y`	X prend la valeur Y

Afficher X
 	
Afficher "abcd"
	`print(X)`
	
	`print("abcd")`
		X peut être à peu près
		de n'importe quel type
		Voir la fonction `print` plus loin.

Si condition Alors
| ...
| ...
FinSi
...
	`if condition:`
	`    ...`
	`    ...`
	`...`
		Les instructions à exécuter si la condition est vraie sont indentées
		
		le retour à l'indentation de départ indique la fin

Sinon Si
 	
Sinon
	`elif condition:`
	`    ...`
	
	`else:`
	`    ...`
		indentation au même niveau que `if`
		
		même chose

Pour I allant de 1 à N Faire
| ...
| ...
Fin Pour
...
	`for I in range(1,N+1):`
	`    ...`
	`    ...`
	`...`
		Les instructions à exécuter dans la boucle sont indentées
		le retour à l'indentation de départ indique la fin
		Voir après les paramètres de `range`

Tant que condition Faire
| ...
Fin Tant que
...
	`while condition:`
	`    ...`
	`...`
		Même principe que pour `if` et `for` pour l'indentation

`range(d,f,p)` génère les entiers de `d` (inclus) à `f` (exclu), avec un pas de `p`. Par défaut, `d=0` et `p=1`.
Exemples : `range(10)` signifie `range(0,10,1)` et génère les entiers de 0 à 9.
`range(2,7)` signifie `range(2,7,1)` et génère les entiers de 2 à 6.
`range(2,9,3)` génère les entiers 2 ;5 ;8.

3. Autres syntaxes Python

a. Opérations sur les nombres

Les opérations élémentaires :

- La somme : `x+y`
- La différence : `x-y`
- Le produit : `x*y`
- La puissance : `x**n` est le résultat de x^n
- La division (décimale) : `x/y`

Il y a deux autres opérations spéciales sur les entiers (nombres de type `int`) liés à la division (euclidienne) entière. Soit `n` et `d` deux entiers. Alors :

- `n%d` est le reste de la division de `n` par `d`.
- `n//d` est le quotient de la division de `n` par `d`. C'est donc un nombre de type `int`.

Autrement dit, si `r=n%d` et `q=n//d`, alors on a l'égalité `n=q*d+r` avec $0 \leqslant$ `r` $<$ `d` si `d` > 0.

b. Tests

Les conditions (dans l'instruction conditionnelle et la boucle conditionnelle) sont des expressions **booléennes**, dont la valeur est `True` ou `False`.

On peut comparer des nombres :

- `x==y` vaut `True` si `x` est égal à `y`. Noter qu'il s'agit d'un « double égal », car l'égalité simple est l'instruction d'affectation et n'est pas un test ;
- `x!=y` vaut `True` si `x` est différent de `y` ;
- `x>=y` vaut `True` si `x` est supérieur ou égal à `y`. De manière similaire, on a `x>y`, `x<=y` et `x<y`.

On peut faire des opérations booléennes, autrement dit composer des tests de la manière suivante :

- `c1 or c2` est `True` si et seulement si `c1` **ou** `c2` est vraie ;
- `c1 and c2` est `True` si et seulement si `c1` **et** `c2` sont vraies ;
- `not c1` est la négation (le contraire) de `c1`.

c. Utilisation de la fonction `print`

La fonction `print` permet d'afficher tout élément affichable (la valeur d'une variable numérique, une chaîne de caractère, etc.)

Elle peut prendre plusieurs arguments. Dans ce cas, par défaut elle va afficher chacun d'eux séparés par un espace et elle termine par un retour à la ligne.

Exemple (on considère ici que `x` vaut 5) :

```
>>> print("abc",3,"4",x)
abc 3 4 5
>>>
```

On peut modifier ce comportement par défaut en ajoutant en paramètre la valeur du caractère de séparation (`sep`) et du caractère de fin (`end`), qui sont en fait des chaînes de caractères. Par défaut, on a `sep=" "` (un espace) et `end="\n"` (un retour à la ligne).

Exemple (`x` vaut encore 5) :

```
>>> print(x,3,"4",sep="/",end="*")
5/3/4*
```

Autre exemple :

```
>>> print(3,"4",sep="",end="*\n*\n")
34*
*
```

d. Fonctions

On définit une fonction, dépendant d'autant de paramètres que l'on veut, de la manière suivante (attention à l'indentation) :

```
def nom_de_fonction(parametre1,parametre2):
    ...
    ...
```

Dans le cas où une fonction ne prend pas de paramètre, on met des parenthèses vides.
Exemple :

```
def affichage_predefini():
    print("ceci est un affichage quelconque")
```

Si l'on souhaite que la fonction retourne une valeur, on doit pour cela utiliser l'instruction **return**.
On pourra alors par exemple utiliser la fonction dans des instructions d'affectation.
Le nom des variables données en argument ou de celle retournée par la fonction n'a pas nécessairement à être le même que dans la définition de la fonction.
Exemple :

```
def aire(r):
    x=3.14*r**2
    return x

rayon=float(input("Donner un rayon : "))
a=aire(rayon)
print("L'aire du disque de rayon",rayon,"vaut",a)
```

Les variables définies dans la fonction sont **locales**, c'est à dire qu'elles n'ont pas d'existence en dehors ou, si elles en ont, c'est qu'elles ont été définies précédemment et n'ont pas nécessairement la même valeur que dans la fonction.
Par défaut, s'il n'y a pas de **return**, une fonction retourne **None** (rien).
L'instruction **return** termine immédiatement l'exécution des instructions de la fonction appelée.
Ainsi, du code écrit à sa suite (avec la même indentation) ne sera donc jamais exécuté.

e. Listes

Une liste est une collection ordonnée d'éléments de types quelconques, pas nécessairement homogène.
On peut y accéder par leur **indice**, qui commence à 0. On peut aussi modifier ses éléments.
Exemples sans explication de quelques utilisations possibles :

```
>>> ma_liste=["a",3,2.1,"bc"]
>>> ma_liste[1]
3
>>> ma_liste[2]
2.1
>>> ma_liste[-4]
'a'
>>> ma_liste[0:2]
['a', 3]
>>> ma_liste[3]=6
>>> ma_liste
['a', 3, 2.1, 6]
```

```
>>> autre_liste=[i**2 for i in range(2,5)]
>>> autre_liste
[4, 9, 16]
>>> 3*['a','b']
['a', 'b', 'a', 'b', 'a', 'b']
>>> liste3=autre_liste+['a','b']
>>> liste3
[4, 9, 16, 'a', 'b']
>>> liste3.append(5**2)
>>> liste3
[4, 9, 16, 'a', 'b', 25]
```

4. Types du langage Python

a. Liste des principaux types

Type	Exemples	Description
`int`	`-3` `7`	Nombres entiers
`float`	`7.0` `3.3` `-5.2`	Nombres flottants (représentation de nombres décimaux)
`str`	`"Hello World!"`	Chaînes de caractères (*string*)
`bool`	`True` `False` `2+3==5` `2>5`	Booléens (numériquement, `True`=1 et `False`=0)
`list`	`[0,6,5.7]` `["a",3,5==2]`	Listes d'éléments (de types non nécessairement identiques)

b. Fonctions sur le typage

Pour vérifier le type d'une expression, on utilise la fonction `type()`.
Exemples :

```
>>> type(5)
<class 'int'>
>>> type([5,3,"a"])
<class 'list'>
```

Il existe aussi des fonctions de conversion :

- Pour convertir une chaîne de caractère en nombre, on utilise les fonctions `int()` et `float()`.
  Exemples :
  ```
  >>> int("5")
  5
  >>> float("5.3")
  5.3
  >>> float("5")
  5.0
  ```
- Pour convertir un nombre en chaîne de caractère, on utilise la fonction `str()`.
  Exemples :
  ```
  >>> str(35)
  '35'
  >>> str(5-9.4)
  '-4.4'
  ```

À savoir : la fonction `input()` retourne une chaîne de caractère. C'est pour cela que, lorsque l'on demande un entier, ou un flottant, on doit l'utiliser en combinaison avec `int()` ou `float()` (voir le tableau de traduction en langage Python précédent).

5. Pour aller plus loin

Beaucoup de fonctions et méthodes sont associées aux éléments de type `str` et `list` entre autres. Pour en savoir plus, consulter la librairie Python 3.