Les structures algorithmiques + implémentation en Python

Le langage de programmation choisi pour implémenter les solutions algorithmiques est le langage de programmation Python.

A. Introduction générale

Python est un langage de programmation sensible à la casse.

Dans un code Python, il est important d'ajouter un commentaire en le précédent par le symbole **#**.

B. Les syntaxes des structures algorithmiques

1. Les opérations élémentaires simples

a. *L'opération d'entrée*

En algorithmique	En python	Remarques
Lire (Objet)	Objet = **input()** Objet = **input(**'message'**)**	La valeur saisie est de type chaîne de caractère (**str**).

b. *L'opération de sortie*

En algorithmique	En python
Écrire ("Message", Objet, Expression)	**print ("Message", Objet, Expression)**
Écrire_nl ("Message", Objet, Expression)	**print ("Message", Objet, Expression, "\n")**
Remarques	
1) Objet est de type simple (entier, réel, booléen, caractère et chaîne de caractères). **2) "\n**", permet d'ajouter un retour à la ligne	

N.B. : L'affichage d'un tableau T en python doit se faire élément par élément et non pas avec l'instruction **print(T)**

c. *L'opération d'affectation*

En algorithmique	En python	Remarques
Objet ← Expression	Objet = Expression	Objet est de type **simple**

2. Les types de données simples

En algorithmique	En python	Exemples
Entier	int	x = int("3") ➔ x reçoit l'entier 3
Réel	float	x = float("3.2") ➔ x reçoit le réel 3.2
Booléen	bool	x = bool("0") ➔ x reçoit True
Caractère	str	x = str(3) ➔ x reçoit le caractère "3"
Chaîne de caractères	str	x = str(123) ➔ x reçoit la chaîne "123"

3. Les structures de données

En algorithmique	En python
Tableau	Ce type sera présenté ci-après

4. Les déclarations

a. Les objets de type de donnée simple

En algorithmique

Objet	Type / Nature
Nom_objet	**Type_objet**

En Python

Une variable **n'a pas besoin d'être déclarée** avec un type particulier : c'est au moment où on lui attribue une valeur qu'elle sera créée et son type sera défini selon le type de la valeur attribuée. Les **noms** des variables sont **sensibles à la casse**.

b. Les tableaux

En algorithmique

1ère formulation **TDO**

Objet	Type / Nature
Nom_tableau	**Tableau de N Type _élément**

2 ème formulation

TDNT

Nouveau type
Type_tableau= Tableau de N Type _élément

TDO

Objet	Type / Nature
Nom_tableau	**Type_tableau**

En Python

- On utilisera la bibliothèque **Numpy** pour implémenter les tableaux.
- Les tableaux numpy sont **homogènes**, c'est-à-dire constitués d'éléments du **même type**.
- Les tableaux **numpy sont statiques** et leurs tailles sont **fixées lors de la création**.
- En Python, la déclaration d'un tableau se fait en deux étapes :
    - L'importation de la bibliothèque **Numpy**
    - La déclaration du tableau

Importation de la bibliothèque	Déclaration
from numpy import array **ou bien** **import numpy as np**	T **= array ([**Type_élément**] *** N**)** **ou bien** T **= array ([**valeur_initiale**] *** N**)**

Remarques :

- ***On peut aussi initialiser les éléments d'un tableau avec la syntaxe suivante :***

  Nom_tableau ***= array ([****Valeur_initiale****] * N,*** *Type_élément****)***

- ***On peut spécifier le type de chaque élément du tableau avec la syntaxe :***

  Nom_tableau ***= array ([****Valeur_initiale****] * N, dtype=****Type_élément****)***

Exemples de déclarations de tableaux

Exemples de déclarations en Python	
from numpy import * T = array ([0] * 20)	Pour déclarer un tableau de 20 cases et l'initialiser par 20 « zéro »
from numpy import * T = array ([float ()] * 100)	Pour déclarer un tableau de 100 réels
from numpy import * T = array ([str] * 50)	Pour déclarer un tableau de 50 chaînes de caractères
from numpy import * T = array ([str()] * 40)	Pour déclarer un tableau de 40 chaînes contenant chacune un seul caractère.
from numpy import * T = array ([''] * 10,dtype='U20')	Pour déclarer un tableau de 10 cases comportant chacune 20 caractères au maximum et l'initialiser par des chaînes vides

5. Les structures de contrôle conditionnelles

En algorithmique	En python
Si Condition **Alors** Traitement **FinSi**	**if** Condition **:** Traitement
Si Condition **Alors** Traitement1 **Sinon** Traitement2 **FinSi**	**if** Condition **:** Traitement1 **else :** Traitement2
Si Condition1 **Alors** Traitement1 **SinonSi** Condition2 **Alors** Traitement2 …… …… …… **SinonSi** conditionN-1 **Alors** TraitementN-1 [**Sinon** TraitementN] **FinSi**	**if** Condition1 **:** Traitement1 **elif** Condition2 **:** Traitement2 …… …… …… **elif** ConditionN-1 **:** TraitementN-1 **else:** TraitementN
Selon<Sélecteur> Valeur1_1**[**, Valeur1_2, …**]** : Traitement1 Valeur2_1 . . Valeur2_2 : Traitement2 …….. **[Sinon**TraitementN**]** **Fin Selon**	**A partir de la version 3.10** **match** Sélecteur: **case** Valeur1 : Traitement1 **case** Valeur2_1 **|** Valeur2_2**:** Traitement2 **case** Sélecteur if Valeur3_1 <=Sélecteur<= Valeur3_2 : Traitement3 case _: TraitementN **N.B. : A utiliser uniquement avec un sélecteur de type scalaire**

6. Les structures de contrôle itératives

a. *La structure de contrôle itérative complète*

En algorithmique
Pour Compteur de **Début** à **Fin [Pas = valeur_pas] Faire** Traitement **Fin Pour**

En Python
For Compteur **in range (**Début **,** Fin+1**,** Pas**) :** **Traitement** **N.B. :** La valeur finale du compteur est exclue de la boucle

Remarques :

- La valeur du **pas** peut être **positive ou négative**. Par défaut, elle est égale à **1**.
- **for** compteur **in range(10)** se lit : pour chaque valeur du compteur **allant de 0 à 9**
- **for** compteur **in range (5, 10)** se lit : pour chaque valeur du compteur **allant de 5 à 9**
- **for** compteur **in range (5, 10, 2)** se lit : pour chaque valeur du compteur **allant de 5 à 9** avec un **pas = 2**
- **for** compteur **in range (9, 0, -1)** se lit : pour chaque valeur du compteur **allant de 9à 1**avec un **pas = -1**
- Ne pas utiliser l'instruction **break** pour forcer l'arrêt de la boucle **for**

b. *Les structures de contrôle itératives à condition d'arrêt*

En algorithmique	
Tant que Condition **Faire** Traitement **Fin Tant que**	**Répéter** Traitement **Jusqu'à** Condition d'arrêt

En Python	
while Condition: **Traitement**	**Initialisation** **while Condition non d'arrêt :** **Traitement**

7. Les modules

a. La déclaration

En algorithmique	En Python
FonctionNom_fonction(pf_1: $type_1$,pf_2:$type_2$, … , pf_n:$type_n$) **: Type_résultat** **DEBUT** Traitement **Retourner**résultat **FIN**	**def Nom_module (pf_1, pf_2, … ,pf_n) :** **Traitement** **Return résultat** **N.B. :** Utiliser return pour retourner **un seul résultat de type simple**
ProcédureNom_procédure(pf_1:$type_1$,pf_2:$type_2$, … , pf_n:$type_n$) **DEBUT** Traitement **FIN**	**def Nom_module (pf_1, pf_2, … ,pf_n) :** **Traitement**

b. L'appel

En algorithmique	En Python
Objet ←Nom_fonction (pe1, …, pen)	**Objet = Nom_module (pe_1, …, pe_n)**
Nom_procédure (pe1, … ,pen)	**Nom_module (pe_1, …, pe_n)**

N.B. : Une **fonction** retourne **un seul résultat de type simple**

c. Le mode de passage

En algorithmique	En Python
Si le mode de passage est par référence (par adresse), on ajoutera le symbole **@** avant le nom du paramètre. **Procédure** Nom_procédure (@pf_1:$type_1$, @pf_2: type2, … , pf_n: $type_n$) **DEBUT** Traitement **FIN**	En python, le paramètre de type **tableau** est, par défaut passé par référence. Nom_module (pf1, pf2,… , pfn) : Traitement

8. Les opérateurs arithmétiques et logiques

a. Opérateurs arithmétiques

Opération	Opérateur en algorithmique	Opérateur en Python
Somme	**+**	**+**
Soustraction	**-**	**-**
Multiplication	*****	*****
Division	**/**	**/**
Division entière	**Div**	**//**
Reste de la division entière	**Mod**	**%**

b. Opérateurs de comparaison

Opération	Opérateur en algorithmique	Opérateur en Python
Egal	**=**	**==**
Différent	**≠**	**!=**
Strictementsupérieur	**>**	**>**
Supérieur ou égal	**≥**	**>=**
Strictementinférieur	**<**	**<**
Inférieur ou égal	**≤**	**<=**
Appartient (**entier**, **caractère**)	**∈**	**in**

c. Opérateurs Logiques

Opération	Opérateur en algorithmique	Opérateur en Python
Négation	**Non**	**not**
Conjonction	**Et**	**and**
Disjonction	**Ou**	**or**

9. Les fonctions prédéfinies

a. Les fonctions sur les types numériques

Algorithmique	Python	Rôle	Exemples
Arrondi(d)	round(d)	Retourne un nombre qui est la valeur de d arrondie à la plus proche valeur.	**Arrondi (5.2)=5** **Arrondi (8.7)=9** **Arrondi (-2.3)=-2** **Arrondi (-4.6)=-5**
RacineCarré(d)	**from math import *** sqrt(d)	Retourne la racine carrée de d si d est positif, sinon elle provoque une erreur.	**RacineCarré(9)=3**
Aléa (vi, vf)	**from random import *** randint(vi, vf)	Retourne un entier aléatoire de l'intervalle [vi, vf].	**Aléa(1,6) donne un entier entre 1 et 6**
Ent (x)	int(x)	Retourne la partie entière de x.	**Ent(5.7)=5** **Ent (-3.2)=-3**
Abs(x)	abs(x)	Retourne la valeur absolue de x	**Abs(-5)=5**

b. Les fonctions sur le type caractère

Algorithmique	Python	Rôle
Ord (c)	**ord** (c)	Retourne le code ASCII du caractère c.
Chr (d)	**chr** (n)	Retourne le caractère dont le code ASCII est d.

c. Les fonctions sur le type chaînes de caractères

Algorithmique	Python	Rôle
Long(ch)	**len(ch**)	Retourne le nombre de caractères de la chaîne ch.
Pos(ch1, ch2)	ch2**.find** (ch1)	Retourne la première position de la chaîne ch1dans la chaîne ch2.
Convch(x)	**str** (x)	Retourne la conversion d'un nombre x en une chaîne de caractères.
Estnum(ch)	ch.**isdecimal**()	Retourne Vrai si la chaîne ch est convertible en une valeur numérique, elle retourne Faux sinon.
Valeur (ch)	**int** (ch) | **float** (ch)	Retourne la conversion d'une chaîne ch en une valeur numérique, si c'est possible.
Sous_chaine(ch, d, f)	ch**[d:f]**	Retourne une partie de la chaîne ch à partir de la position d jusqu'à la position f (f exclue).
Effacer (ch, d, f)	ch = ch**[: d]+ch[f :]**	Efface des caractères de la chaîne ch à partir de la position d jusqu'à la position f (f exclue).
Majus(ch)	ch.**upper**()	Convertit la chaîne ch en majuscules

N.B. : On utilise l'opérateur **+** pour concaténer deux chaînes.