

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 recommandé d'ajouter des commentaires.
 - Commentaire sur une seule ligne : débuter la ligne par le symbole #.
 - Commentaire sur plusieurs lignes : délimiter les lignes du commentaire par """.

B. Les syntaxes des structures algorithmiques

1. Les opérations élémentaires simples

a. L'opération d'entrée

En algorithmique	En python
Lire (Objet)	Objet = input() Objet = input('message') N.B. : Par défaut, la valeur saisie est de type chaîne de caractères.

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 :

- Objet est une variable de type variable simple (entier, réel, booléen, caractère et chaîne de caractères).
- "\n" permet d'ajouter un retour à la ligne.
- 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
Objet ← Expression	Objet = Expression

Remarque : Objet est une variable de type simple (entier, réel, booléen, caractère et chaîne de caractères).

Les types de données simples

En algorithmique	En python
Entier	int
Réel	float
Booléen	bool
Caractère	str
Chaîne de caractères	str

Exemples de conversions entre les types simples en python

Conversion	Syntaxe	Exemple
De str vers int	int(ch)	x = int("3") signifie que x reçoit l'entier 3
De str vers float	float(ch)	x = float("3.2") signifie que x reçoit le réel 3.2
De str vers bool	bool(ch)	x = bool("0") signifie que x reçoit True
De int vers str	str(int)	x = str(3) signifie que x reçoit le caractère "3"
		x = str(123) signifie que x reçoit la chaîne "123"

3. Les structures de données

En algorithmique	En python
Tableau (à une et à deux dimensions)	Ces types seront présentés ci-après (voir 4.b), 4.c) et 4.d).
Enregistrement	
Fichier	

4. Les déclarations

a. Les objets de type de donnée simple

En algorithmique				
<table border="1"> <thead> <tr> <th>Objet</th> <th>Type / Nature</th> </tr> </thead> <tbody> <tr> <td>Nom_objet</td> <td>Type_objet</td> </tr> </tbody> </table>	Objet	Type / Nature	Nom_objet	Type_objet
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. Ainsi, son type sera défini en fonction du type de la valeur qui lui a été attribuée. L'identificateur d'une variable est sensible à la casse.				

b. Les tableaux

En algorithmique

	Objet	Type / Nature
Tableau à une dimension	Nom_tableau	Tableau de N Type _élément
Tableau à deux dimensions	Nom_tableau	Tableau de N lignes * M colonnes Type _élément

En Python

- On utilisera la bibliothèque **numpy** pour implémenter les tableaux.
N.B. : Exceptionnellement pour l'année scolaire 2021-2022, on acceptera l'utilisation du type « list » uniquement pour le niveau 4^{ème} année.
- Un tableau de la bibliothèque numpy est :
 - homogène, c'est-à-dire constitué d'éléments de même type,
 - statique, car sa taille est fixée lors de la création.
- La déclaration d'un tableau se fait en deux étapes :
 - Importation des modules nécessaires de la bibliothèque **numpy**

Importation

```
from numpy import array  
ou  
from numpy import *  
ou  
import numpy as alias
```

- Déclaration du tableau

Déclaration

T = array ([Type_élément] * N)
ou bien
T = array ([valeur_initiale] * N)

T = array ([Type_élément]*Colonnes]* Lignes)
ou bien
T = array ([valeur_initiale]*Colonnes)* Lignes)

Tableau à une dimension

Tableau à deux dimensions

Remarque : On peut spécifier le type des éléments d'un tableau avec la syntaxe :

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

Exemples de déclarations de tableaux en Python

Déclaration	Explication
T = array ([5] * 10)	Déclarer un tableau T de 10 entiers et initialiser ses éléments par « 5 ».
T = array ([float ()] * 10)	Déclarer un tableau T de 10 réels et initialiser ses éléments par « 0.0 ».
T = array ([str] * 10)	Déclarer un tableau T de 10 chaînes de caractères.
T = array ([str()] * 10)	Déclarer un tableau T de 10 caractères et initialiser ses éléments par le caractère vide.
T = array ([''] * 10 , dtype = 'U20')	Déclarer un tableau T de 10 éléments initialisés par une chaîne vide. Chaque élément peut contenir 20 caractères au maximum.
T = array ([[int ()] * 10]*30)	Déclarer un tableau T de 30 lignes x 10 colonnes d'entiers.

c. L'enregistrement

En algorithmique	
Objet	Type / Nature
Nom_enregistrement	Enregistrement Nom_champ1 : Type_champ1 Nom_champ2 : Type_champ2 ... Fin
En Python	
<pre>Nom_enregistrement = dict (Nom_champ1 = Type_champ1, Nom_champ2 = Type_champ2, ...)</pre>	
Remarque : Pour accéder à un champ d'un enregistrement on utilise la syntaxe suivante : Nom_Enregistrement ['Nom_Champ'].	

a. Les fichiers

En algorithmique		
	Objet	Type / Nature
Fichier texte	Nom_fichier	Fichier Texte
Fichier de données	Nom_fichier	Fichier de Type _élément
En Python		
La déclaration d'un objet de type fichier se fait lors de sa création à l'aide de la fonction open() détaillée ci-après (voir 10.a) et 10.b)).		

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 Sinon Si Condition2 Alors Traitement2 [Sinon TraitementN] FinSi	if Condition1 : Traitement1 elif Condition2 : Traitement2 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 V3_1 <= Sélecteur <= V3_2 : Traitement3 case _ : TraitementN

N.B. : Le sélecteur doit être de type scalaire.

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.
- 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	En Python
Tant que Condition Faire Traitement	
Fin Tant que	while Condition :
Répéter Traitement	Traitement
Jusqu'à Condition d'arrêt	

Remarque : Ne pas utiliser l'instruction **break** pour forcer l'arrêt de la boucle **while**. Toutefois, pour l'année scolaire 2021/2022 on acceptera l'implémentation python de la boucle **Répéter** via **while True ... break**

Les modules

a. La déclaration

En algorithmique	En Python
Fonction Nom_fonction (pf ₁ : type ₁ , pf ₂ : type ₂ , ..., pf _n : type _n): Type_résultat DEBUT Traitement Retourner résultat FIN	Un module (fonction ou procédure) se définit en utilisant le mot clé def selon la syntaxe suivante : <pre>def Nom_module (pf₁, pf₂, ..., pf_n): Traitement [return résultat]</pre>
Procédure Nom_procédure (pe ₁ : type ₁ , pe ₂ : type ₂ , ..., pe _n : type _n) DEBUT Traitement FIN	<i>N.B. :</i> Dans un module, l'instruction "return" peut être utilisée, et ce, pour retourner un seul résultat de type simple .

b. L'appel

Module	En algorithmique	En Python
Fonction	Objet ← Nom_fonction (pe ₁ , ..., pe _n)	Objet = Nom_module (pe ₁ , ..., pe _n)
Procédure	Nom_procédure (pe ₁ , ..., pe _n)	Nom_module (pe ₁ , ..., pe _n)

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 : type ₂ , ..., pf _n : type _n) DEBUT Traitement FIN	<pre>def Nom_module (pf₁, pf₂, ..., pf_n): Traitement</pre> <i>N.B. :</i> En python, les paramètres de type dictionnaire , tableau et fichier sont, par défaut passés par référence.

d. La portée des variables en python :

- Toute variable déclarée au sein d'un module a une **portée locale**.
- Toute variable déclarée au sein d'un module précédée par le mot clé **global** a une **portée globale**. Par conséquent, elle ne devra pas figurer parmi les paramètres de ce module.

Exemple d'un programme en python présentant une solution modulaire

```
from numpy import array
T1 = array([ ]* ) #Déclaration du tableau T1
T2 = array([ ]* ) #Déclaration du tableau T2
#définition du module saisieTaille
def saisieTaille (bornInf , bornSup) :
    taille =
        while taille not in range(bornInf, bornSup+ ):
            taille=int(input("Taille entre "+str(bornInf)+" et "+str(bornSup)+" :"))
    return taille
#définition du module remplirTab
def remplirTab (T, taille) :
    for i in range( taille ) :
        T[i] = int( input ("Donner l'élément N° " + str(i) + " : "))
#définition du module afficherTab
def afficherTab (T , taille) :
    for i in range(taille) :
        print(T[i])
#le programme principal
n = saisieTaille ( , --) #1er appel du module saisieTaille
m = saisieTaille ( , ) #2ème appel du module saisieTaille
print("chargement de T1")
remplirTab(T1 , n)
print("chargement de T2")
remplirTab(T2 , m)
print("Affichage du tableau T1")
afficherTab(T1 , n)
print("Affichage du tableau T2")
afficherTab(T2 , m)
```

2^{ème} façon d'implémentation du module saisieTaille en utilisant une variable globale nommée Taille

```
#définition du module saisieTaille
def saisieTaille (bornInf , bornSup) :
    global taille
    taille =
        while taille not in range( bornInf , bornSup+ ) :
            taille=int(input("Taille entre "+str(bornInf)+" et "+str(bornSup)+" :"))
#Le programme principal
saisieTaille ( , ) #1er appel du module saisieTaille
n = taille
saisieTaille ( , ) #2ème appel du module saisieTaille
m = taille
print("chargement de T1")
remplirTab(T1 , n)
print("chargement de T2")
remplirTab(T2 , m)
print("Affichage du tableau T1")
afficherTab(T1 , n)
print("Affichage du tableau T2")
afficherTab(T2 , m)
```

Les opérateurs arithmétiques et logiques

a. Opérateurs arithmétiques

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

b. Opérateurs de comparaison

Opération	En algorithmique	En Python
Egal	=	==
Different	≠	!=
Strictement supérieur	>	>
Supérieur ou égal	≥	≥
Strictement inférieur	<	<
Inférieur ou égal	≤	≤
Appartient (entier, caractère)	ε	in

c. Opérateurs logiques

Opération	En algorithmique	En Python
Négation	Non	not
Conjonction	Et	and
Disjonction	Ou	or

9. Les fonctions prédéfinies

a. Les fonctions sur le type numérique

En algorithmique	En Python	Observation
Arrondi (x)	round (x)	
RacineCarré(x)	sqrt (x)	Nécessite l'importation de la bibliothèque math .
Aléa (vi, vf)	randint(vi, vf)	Nécessite l'importation de la bibliothèque random .
Ent(x)	int (x)	
Abs (x)	abs (x)	

b. Les fonctions sur le type caractère

En algorithmique	En Python
Ord (c)	ord (c)
Chr (d)	chr (d)

c. Les fonctions sur le type chaîne de caractères

En algorithmique	En Python
Long(ch)	len(ch)
Pos(ch1, ch2)	ch2.find (ch1)
Convch(x)	str (x)
Estnum(ch)	ch.isdecimal()
Valeur (ch)	int (ch) float (ch)
Sous_chaine(ch, d, f)	ch[d:f]
Effacer (ch, d, f)	ch = ch[: d]+ch[f :]
Majus(ch)	ch.upper()

Remarque : Pour concaténer deux chaînes de caractères, on utilise l'opérateur « + ».

10. Les fonctions et les procédures prédéfinies sur les fichiers

a. Les fichiers de données

En algorithmique	En Python
Ouvrir("Chemin\Nom_physique", Nom_logique , "Mode") Avec mode d'ouverture égal à : <ul style="list-style-type: none"> o "rb" : Lecture (pointer au début) o "wb" : Ecriture (création) o "ab" : Ajout à la fin du fichier 	Nom_logique=open (‘Chemin\Nom_physique’ , ‘Mode’)
Lire (Nom_logique , Objet)	from pickle import load, dump Objet = load (Nom_logique)
Ecrire (Nom_logique , Objet)	from pickle import load, dump dump (Objet , Nom_logique)
Fin_fichier (Nom_logique)	Fin_fichier = False while not (Fin_fichier) : try : x = load (Nom_logique) except : Fin_fichier = True
Fermer (Nom_logique)	Nom_logique.close ()

b. Les fichiers textes

En algorithmique	En Python
Ouvrir ("Chemin\Nom_physique" , Nom_logique , "Mode") Avec mode d'ouverture égal à : <ul style="list-style-type: none"> o "r" : Lecture o "w" : Ecriture (création) o "a" : Ajout à la fin du fichier 	Nom_logique = open ("Chemin\Nom_physique" , 'Mode')
Lire (Nom_logique , ch)	ch = Nom_logique.read()
Lire_ligne (Nom_logique , ch)	ch = Nom_logique.readline()
Ecrire (Nom_logique , ch)	Nom_logique.write(ch)
Ecrire_nl (Nom_logique , ch)	Nom_logique.write(ch + "\n")
Fin_fichier (Nom_logique)	ch= Nom_logique.readline() While ch != "" : Traitement ch = Nom_logique.readline() <i>N.B. : La fin d'un fichier texte est la chaîne vide</i>
Fermer (Nom_logique)	Nom_logique.close ()

Remarque : Lors de la résolution d'un problème, il est fortement **interdit d'utiliser autres fonctions ne figurant pas dans la liste des fonctions énumérées** dans le présent document. Toutefois, les énoncés des épreuves pratiques du baccalauréat des matières « Informatique » et « Algorithmique et programmation », pourraient intégrer une **nouvelle fonction**. Dans ce cas, le rôle et la **syntaxe** de cette fonction seront détaillés dans l'énoncé de l'épreuve.