3. Correspondances algo/Python

Voici un tableau qui va vous présenter la correspondance entre les syntaxes algorithmiques et le langage Python.

Algo	Python
	`# un commentaire`

`nomVariable : typeVariable`
exemple :
`k : entier`
	`nomVariable = valeur`
	`# le typage est implicite`

`nomTableau[1..N] : typeTableau`
exemple :
`tab[1..10] : entier`
accéder à une case :
`tab[indice]`
	`nomTableau = [valeur1, ..., valeurN]`
	`# le tableau est défini au moment de l'initialisation : le typage est implicite`
	`# Il est aussi possible d'initialiser avec la même valeur :`
	`nomTableau = [valeur]*taille`
	`# Accéder à une case :`
	`tab[indice]`
	`(voir plus bas plus d'informations sur les tableaux)`

`nomVariable ← valeur`	`nomVariable = valeur` `var1 = var2 = valeur`
signes de comparaison et logique : `=, <>, <, >, <=, >=, non, et, ou, dans`	`==, !=, <, >, <=, >=, not, and, or, in`
opérateurs logiques : `et, ou, oux, ~ (inverse)`	`&, |, ^, ~`
opérations : `+, -, *, /, div, mod`	`+, -, *, /, //, %`

Quelques raccourcis d'écriture :
`A ← A + B`
`A ← A – B`
`A ← A * B`
`A ← A / B`
	`A += B`
	`A -= B`
	`A *= B`
	`A /= B`

`si condition alors`
`  action`
`finsi`
	`if condition :`
	`  action1`

`si condition alors`
`  action1`
`sinon`
`  action2`
`finsi`
	`if condition :`
	`  action1`
	`else :`
	`  action2`

`si condition1 alors`
`  action1`
`sinonsi condition2 alors`
`  action2`
`sinon`
`  action3`
`finsi`
	`if condition1 :`
	`  action1`
	`elif condition2 :`
	`  action2`
	`else :`
	`  action3`

tantque condition
  action
fintantque
	while condition :
	  action

repeter action jusqu'à condition	

pour k de deb à fin
  action
finpour
	for k in range(deb, fin+1) :
	  action
	# remarque : voir la fonction de range dans les fonctions prédéfinies, plus bas

pour chaque element dans liste
  action
finpour
	for element in liste :
	  action
	# remarque 1 : la liste peut être une chaine (liste de caractères), dans ce cas l'élément est un caractère.
	# remarque 2 : in peut être utilisé dans un if.

procedure nomProc (param1 : type1, …, paramN : typeN)
  déclarations
debut
  traitements
fin
	def nomProc (param1, ..., paraN) :
	  """commentaire officiel"""
	  traitements
	# remarque 1 : les paramètres peuvent recevoir une valeur par défaut.
	# remarque 2 : lors de l'appel, il est possible de préciser le nom du paramètre, ce qui permet de changer l'ordre des paramètres : nomProc(param2=valeur...)
	# remarque 3 : le commentaire officiel (docstring) est optionnel, mais s'il est présent, il est reconnu comme commentaire du module et peut être récupéré avec la fonction help.

fonction nomFonc (…) : typeRetour
  déclarations
debut
  traitements
  retourner valeur
fin
	def nomFonc (param1, ..., paraN) :
	  """commentaire officiel"""
	  traitements
	  return valeur
	# Remarque : il existe une écriture possible pour des fonctions d'une seule instruction.
	variable=lambda param1, ..., paramN : instruction
	# Remarque : le mot lambda est réservé et obligatoire. La fonction va retourner le résultat de l'instruction.
	# Voici comment appeler cette fonction :
	variable(valeur1, ..., valeurN)
	# Remarque : attention cela reste une fonction, mais dont le nom sera la variable, donc il faut l'utiliser comme une fonction (par exemple en l'afichant)

pour écrire une instruction sur plusieurs lignes (pas d'équivalence en algo)	`une instruction \` `... la suite à la ligne suivante`

pour sortir d'une boucle ou remonter au début de la boucle (pas d'équivalence en algo)	`# deux instructions à éviter :`
	`break # pour sortir d'une boucle`
	`continue # retourne au début de la boucle`

4. Compléments sur les variables

Au-delà des informations données dans le tableau précédent, voici quelques compléments sur les variables.

Nommage des variables et modules

Les noms des variables (et des modules) doivent être uniques.

Un nom doit respecter les règles suivantes :

- il peut être composé de lettres, chiffres et underscores (rien d'autre)
- il ne peut commencer par un chiffre
- il ne peut être identique à un mot réservé du langage
- attention à la casse (la variable AGE est différente de la variable Age)

Types des variables

Python gère la déclaration à la volée. Les types sont donc implicites

Voici cependant les types qu'il reconnait :

nom du type	explication	exemple
int	entier	3
float	flottant (décimal)	3.2 remarque : le point est le séparateur décimal
str	chaîne	"bonjour" 'bonjour' """bonjour""" "j'aime le Python" 'j\'aime le Python'
bool	booléen	True False

Déclaration d'une constante

Il n'y a pas de notion de constante : il suffit d'affecter une valeur à une variable.

Déclarations des variables globales

Une variable initialisée en dehors des modules est globale. Mais attention, pour qu'elle soit reconnue dans un module, il faut rappeler son utilisation dès le début du module :

```
# déclaration globale
maVar = 15
# utilisation dans un module
def unModule() :
   global maVar
   print(maVar) # on obtient bien 15
```

Déclarations de variables locales

Une variable initialisée dans un module est locale au module.

Les tableaux

En Python, les tableaux sont en réalité des listes de valeurs. Ces valeurs peuvent ne pas être de même type.

Différentes méthodes de déclaration d'une liste :

```
# liste vide
liste = []
# liste initialisée avec une valeur (et en fixant la dimension)
liste = [valeur] * dimension
# liste contenant des listes (tableau à plusieurs dimensions)
liste = [] # création de la liste
liste.append([]) # ajout d'une autre liste dans une des cases
```

Accès à une valeur d'une liste :

```
# récupération d'une seule valeur par son indice :
liste[indice]
# récupération d'un groupe de valeurs :
liste[debut:fin] # les valeurs sont récupérées à partir de l'indice debut
et jusqu'à l'indice juste avant fin
liste[debut:] # les valeurs sont récupérées à partir de l'indice debut
```

Voici quelques méthodes applicables sur les listes :

nom	explication	exemple
sort()	tri de la liste	liste = [25, 12, 43, 2] liste.sort() print(liste) # [2, 12, 25, 43]
append(valeur)	ajoute une valeur en fin de liste	liste = [25, 12, 43] liste.append(14) print(liste) # [25, 12, 43, 14]
reverse()	inverse l'ordre de la liste	liste = [25, 12, 43, 2] liste.reverse() print(liste) # [2, 43, 12, 25]
remove(valeur)	supprime une valeur de la liste	liste = [25, 12, 43, 2] liste.remove(12) print(liste) # [25, 43, 2]

index(valeur)	retourne la position (l'indice) de la valeur passée en paramètre (l'indice commence à 0)	liste = [25, 12, 43, 2] val = liste.index(12) print(val) # 1
pop()	retourne la dernière valeur de la liste	liste = [25, 12, 43, 2] val = liste.pop() print(val) # 2
count(valeur)	retourne le nombre d'occurrences de la valeur passée en paramètre	liste = [25, 12, 43, 2, 12] nb = liste.count(12) print(nb) # 2
extend(autreliste)	ajoute le contenu de autreliste à la suite de la liste.	liste = [5, 12, 3] autreliste = [21, 2] liste.extend(autreliste) print(liste) # [5, 12, 3, 21, 2]

Distinction entre caractère et chaîne de caractères

Une chaine est considérée comme un tableau de caractères (comme une liste). Pour accéder à un caractère de la chaine, il est donc possible de préciser son indice.

Exemple :

```
ch = "Bonjour"
print(ch[3])   # on obtient “j” à l’affichage
```


5. Syntaxe et règles fondamentales

Les indentations

Python impose les indentations dans les blocs d'instructions. Donc, à chaque fois qu'il est conseillé d'indenter dans un algo, il est obligatoire d'indenter en Python.

Le type d'indentation est libre (un ou plusieurs espaces ou une tabulation).

La casse

Python est sensible à la casse.

Donc par exemple, la variable AGE n'est pas la même que la variable Age ou la variable age.

Les encadrements et fins d'instructions

Les débuts de blocs (début de module, de boucle ou de condition) sont marqués par ":".

Il n'y a pas de signe de fin de bloc car ce sont les indentations qui permettent de repérer les blocs.

Les commentaires

Les commentaires sont de 2 types :

- Les commentaires normaux qui commencent par le signe #
- Les commentaires officiels (pour créer le docstring) encadrés par 3" :

```
""" commentaire officiel """
```

Permutation

Pour permuter 2 variables, Python propose un raccourci d'écriture :

exemple :

```
a = 10
b = 23
a,b = b,a
```

Cette syntaxe inverse le contenu des variables a et b.

6. Fonctions prédéfinies

Affichage et saisie

nom fonction	explication	exemple
print(une_chaine)	affiche à l'écran le contenu du paramètre	a = 4 b = 7 print("a = ",a," et b = ",b) on obtient : a = 4 et b = 7
input(un_message)	affiche le message et retourne la valeur saisie au clavier	age = input("donnez votre âge")

Manipulation des types

nom fonction	explication	exemple
type(nom_variable)	retourne le type de la variable passé en paramètre	a = 4 print type(a) on obtient : <class 'int'>

int(nom_variable)	retourne le contenu de la variable au format integer	# saisie age au format string
		age = input("age = ")
		# convertit age en entier
		age = int(age)

float(nom_variable)	idem pour float	
str(nom_variable)	idem pour string	
ord(caractere)	retourne le code ascii d'un caractère	print(ord('a')) # 97
chr(ascii)	retourne le caractère correspondant au code ascii	print(chr(97)) # a

Autres fonctions

nom fonction	explication	exemple
help(nom_module)	Affiche la docstring du module (le commentaire officiel).	
len(variable)	retourne la taille de la variable (nombre de caractères pour une chaine, nombre de cases pour un tableau)	
range(nb) range(min, max) range(min, max, step)	retourne une séquence de valeurs	range(4) # [0, 1, 2, 3] range(3, 6) # [3, 4, 5] range(4, 9, 2) # [4, 6, 8]