Correspondances algo/Python

Voici un tableau qui va vous présenter la correspondance entre les syntaxes algorithmiques et le langage Python.
Pour plus d'explications, aller voir https ://www.w3schools.com/python/default.asp

Algo	**Python**
	`# commentaire`

Entrées/sorties
`Afficher nomVariable`
`Lire nomVariable`
	
	`print(nomVariable)`
	`nomVariable = input()`

Les types : entier flottant chaîne booléen	Python gère la déclaration à la volée. Les types sont donc implicites Voici cependant les types qu'il reconnait : int `# 3` float `# 3.2` str `# "bonjour" ou 'bonjour'` bool `# True False`

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

Affectation : `nomVariable` ← `valeur`	 `nomVariable = valeur`
signes de comparaison et logique : $=, \neq, <, >, \leq, \geq$, non, et, ou	(dans le même ordre :) `==, !=, <, >, <=, >=, not, and, or`
opérateurs logiques bit à bit : et, ou, oux, $\sim$ (inverse)	(dans le même ordre :) `&, |, ^, ~`
opérations : `+, -, *, /, div, mod`	(dans le même ordre :) `+, -, *, /, //, %`

Quelques raccourcis d'écriture :
$A \leftarrow A + B$
$A \leftarrow A – B$
$A \leftarrow A \times B$
$A \leftarrow A/B$
	
	`A += B`
	`A -= B`
	`A *= B`
	`A /= B`

Tableaux 1D :
`nomTableau[1..N] : typeTableau`
exemple :
`nomTableau[1..10] : entier`
accéder à une case :
`nomTableau[indice]`
	
	`nomTableau = [valeur1, ..., valeurN]`
	
	`# accéder à une case :`
	`nomTableau[indice]`

Tableaux 2D :
`nomTableau[1..N, 1..M] : typeTableau`
exemple :
`nomTableau[1..10, 1..3] : entier`
accéder à une case :
`nomTableau[ligne][colonne]`
	
	`nomTableau = [[val11, ..., val1M],`
	`...,`
	`[valN1, ..., valNM]]`
	`# accéder à une case :`
	`nomTableau[ligne][colonne]`

Chaînes de caractères :
`chaine1 <- "hello"`
`chaine2 <- "world"`
`chaine <- chaine1 + " " + chaine2`
`taille(chaine)`
`extraire(chaine, 6, 5)`
`extraire(chaine, 6)`
`x <- position("r", ch1)`
`uneVariableChaine <- str(3)`
`uneVariableNumerique <- val("3")`
 	
Insérer, remplacer...
	`chaine1 = "hello"`
	`chaine2 = "world"`
	`chaine = chaine1 + " " + chaine2`
	`len(chaine)`
	`chaine3[6:11]` (extraire du car. 6 au car. 11)
	`chaine[6:]` (extraire du car. n° 6 jusqu'à la fin)
	`x = chaine.find("r")`
	`uneVariableChaine = str(3)`
	`unEntier = int("3")` (vers un entier)
	`unFlottant = float("3")` (vers un flottant)
	
	Voir la documentation :
	`www.w3schools.com/python/python_strings.asp`

si *condition* **alors**
| action
fin
	`if condition :`
	`  action`
	
	Remarque : attention de bien indenter
	Ce qui n'est plus indenté n'est plus dans le `if`

si *condition* **alors**
| action1
sinon si *condition2* **alors**
| action2
sinon
| action3
fin
	`if condition :`
	`  action1`
	`elif condition2 :`
	`  action2`
	`else :`
	`  action3`

tant que *condition* **faire** | action **fin**	`while condition :` `  action`
répéter | action **jusqu'à** *condition*	
pour *k* ***de*** *deb* **à** *fin* **faire** | action **fin**	`for k in range(deb, fin+1) :` `  action`
pour *k* ***de*** *deb* **à** *fin* ***par*** *pas* ***de*** **faire** | action **fin**	`for k in range(deb, fin+1, pas) :` `  action`
pour *chaque element dans liste* **faire** | action **fin**	`for element in liste :` `  action`
déclarations **début** | traitement **fin** **Procédure** `nomProc(`*param1 : type1, ..., paramN : typeN*`)`	`def nomProc (param1, ..., paraN) :` `  traitements`

déclarations
début
| traitement
| retourner valeur
fin
Fonction `nomFonc(...)` : typeRetour
	`def nomFonc (param1, ..., paraN) :`
	`  traitements`
	`  return valeur`