Langage PYTHON

◀ Types de base

- opérations sur les entiers (`int`) : `+` , `-` , `*` , `//` , `**` , `%`

- opérations sur les flottants (`float`) : `+` , `-` , `*` , `/` , `**`

- opérations sur les booléens (`bool`) : `not`, `or`, `and`

Evaluation paresseuse : `while i<n and L[i]==0 : i = i+1`

Valeurs booléennes : sont identifiés à `False` l'entier 0 et la liste vide []. On identifie souvent 1 et `True`.

Exemple : si P est une pile, on utilise souvent le test : `while P : ...`

- comparaisons : `==` , `!=` , $<$, $>$, `<=` , `>=`

- type intervalle d'entiers : `range(a,b,h)` représente les entiers $x = a + kh$, avec k entier tel que $a \leq x < b$.

◀ Structures de contrôle

- instruction conditionnelle : `if` , `elif` , `else`

- boucle `while`, `break` dans un corps de boucle (on peut utiliser `return` au sein d'une fonction)

- boucle for : Itérateurs : `range(i,j)`, une chaîne, un tuple, une liste, un dictionnaire (`keys` ou `items`).

Exemple : `for i in range(n-1,-1,-1) : instruction(i)`

- définition d'une fonction `def f(x:type) -> type :` ... `return` ...

Exemple : `def f(x:int,y:int) -> bool : return x ==0 or y ==0`

Exemple : `def f(L:list) -> None : L.append(1)`

- distinction entre fonction et procédure (avec un argument mutable).

- adressages identiques et nécessité éventuelle de copies : Différence entre :

`a = [[]]*3` et `a = [[] for i in range(3)]`, avec `a[0].append(1)`

- affectations simultanées : `x,y = y,x` équivaut à `z=x ; x=y ; y=z`

Exemple : Inversion d'une sous-séquence dans une liste :

`L[a:b] = [ L[k] for k in range(b-1,a-1,-1)]`

Remarque : Ne convient pas lorsque `x,y,z` sont mutables.

◀ Structures indicées immuables (= non mutables) : **chaînes, tuples**

Elles doivent être considérées **comme un bloc non modifiable**.

- longueur : `len((1,3,2)) ; len("coucou")`

- accès par indice positif valide : `A = "prepa" ; A[0] ; B = (3,4) ; B[1]`

- concaténation des chaînes : `"prepa" + "llg"`

◀ Structures mutables : listes et dictionnaires

Listes

- longueur et accès : `n = len(L) ; L[i]` où $0 \leq i < n$ accès en temps (amorti) $O(1)$

- accès par indice positif valide ($0 \leq i < n$) et modification d'un élément : `L[i] = x`

- création par compréhension : `[f(i) for i in J]`

- structure de pile : `L.append(x) ; x = L.pop()`

Exemple :
```
def inverse(L) :
    M = []
    for x in L : M.append(x)
    L = M
```

- concaténation des listes : `[0,1,2]+[3,4] ; [0]*5`

- tranches : `A[i:j]` (crée une nouvelle liste via de nouvelles cases mémoires)

- copie : `M = L.copy()` ou bien `M = L[:]` ou bien `M = list(L)`

Remarque : ne pas confondre avec `copy(L)` du module `numpy` qui renvoie un vecteur (`array`).

C*aractère superficiel des copies* :

`L = [ [0,0], [0,0] ] ; M = L.copy() ; L[0][0] = 1`

`M[0][0]` vaut à la sortie `1` car `L[0]` et `M[0]` pointent vers le même objet.

Dictionnaires

`dico_vide = { } ; dico = {1 : 30 , 2 : 20 , 3 : 10}`

`dico = {'a' : 3 , 'b' : 2 , 'c' : 1} ; dico['a'] ; len(dico)`

Ajout : `dico['e'] = 3 ;` suppression d'une clé : `del dico['a']`

Appartenance : `(c in dico)` renvoie `True` ssi c est une clé.

Itération : `for c in dico` ... permet d'itérer sur les *clés* ; variante : `for c in dico.keys()`

`for v in dico.values()` ... permet d'itérer sur les *valeurs* d'un dictionnaire.

`for k,v in dico.items()` permet d'itérer sur les couples (*clé,valeur*) d'un dictionnaire.

◂ Dépaquetage des listes et des tuples

Exemple : `L = [1,5] ; [i,j] = L ;`

Exemple : `T = (4,3,2) ; (x,y,z) = T  # variante : x,y,z = T`

◂ Divers

- type `None` : PYTHON propose le type *NoneType*, dont la seule valeur est `None` : c'est une représentation de l'absence de valeur. L'oubli de `return` est une erreur classique. Dans ce cas, la fonction renvoie la valeur `None`.

- utilisation `print(...);` un affichage obtenu par `print` dans le corps d'une fonction ne doit pas être considéré comme le résultat de la fonction (ne pas confondre avec `return`) : c'est un simple effet de bord). D'ailleurs la fonction `print` renvoie la valeur `None`.

- importation de modules : `import module, import module as alias, from module import *`

- vérifications : `assert test` (culturel : ou bien `assert test,message`).

`def modif(L,i) : assert (i>=0 and i<len(L)) ; ...`

Interruption de la fonction si le test vaut `False` (et affichage d'un message éventuel).