# Algorithmique et programmation structurée orientée Python

• Python :

— interprété

— multiparadigme : on peut utiliser plusieurs paradigmes de programmation (fonctionnel, orienté objet, impératif)

— multiplateforme

• Syntaxe :

— langage indenté, sans accolade

— les commentaires sont précédés du symbole #

— la docstring (documentation) se situe au-dessus de la fonction (obligatoire)

— les variables sont déclarées sans type, le type étant déterminé automatiquement à l’affectation d’une valeur

— il faut donner des noms parlants aux variables

> déclarer une variable

— opérations arithmétiques

— déclarer une fonction avec le mot-clé def :

Docstring

Def fonc(param1, param2):

— boucles for et while

• Bloc de code : déterminé par l’indentation

• fonction help() :

def add(a,b):

…

help(add) # affiche la documentation de la fonction add

• variables déclarées sans type

• types entier, flottant, chaine de caractères

• variable : référence vers un espace mémoire

• Les types sont des classes.

Liste list

Tuple

Dict doct ( 1 : "a"

Set ensemble {1

• méthode (utilisation) : a.upper() ; lower(), capitalize, title, swapcase, count, find

• Concaténation : c = a + b

• Répéter une chaine de caractère : b = a \* 3

• Indexer et tronquer chaine

• Formater chaine :

a = "Bonjour {0}, comment vas-tu ?".format("Jean")

prenom = "Pierre"

b = f"Bonjour {prenom}, comment vas-tu ?"

a = "{0}, {1}, {2}".format("Jean","Pierre","Paul")

• Conversion : python peut convertir automatiquement les types dans la limite du possible

On va faire du cast ou transtypage : int(variable), float(variable)

• type(variable) : récupérer le type de la variable

• clear = pour effacer le terminal

Structures de données :

• Listes :

— []

— avec un système d’index

— le premier élément de la liste est à 0

— mutable : on peut avoir différents types dans une liste

— on peut modifier les éléments d’une liste

— append, clear, len, count

• Tuples :

— immutables

— doublons

• Set :

— pas de doublon

— très rapide pour l’insertion de données, lent pour le reste

— pas d’index

Set = {1, 2, 3, 4, 5}

Set.add(élément)

— Pour utiliser les index -> transformer en liste

• Dictionnaire = prend une forme de clé/valeur

mutables

Dic = {

"nom" : "Dupont",

"age" : 25

}

On accède aux éléments par la clé, pas l’index

Dicto['age'] = 26

Dico.clear()

.values

.items

.pop

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Listes | Tuples | Set | Dictionnaires |
| Avantages | • mutables  • ordonnées | • immuables  • ordonnées  • rapides  • immodifiables | • mutables  • pas de doublons | • mutables  • rapides  • clé/valeur  • ordonnés (depuis Python 3.7) |
| Inconvénients | • lentes  • doublons  • index |  |  | • pas ordonnés (avant Python 3.7)  • plus lents que listes et les tuples en insertion et suppression |

Structures de contrôle

Conditions (if)

Boucles (while)

Itérative (for)

If a is b : is vérifie si a et b pointent vers le même objet

Dic : parcourir keys / values / items

Lamda expression : expression anonyme, qui n’a pas de nom, que l’on ne va jamais réutiliser

Star = lambda i : "\*" \* i

For i in range(1, nombre+1):

Print(start(i))

• Fonction : morceau de code réutilisable

• Factorisation : réduire le nombre de lignes de code, de répétitions, de variables, etc. dans une volonté de rendre le code plus lisible et maintenable

• Déclaration : def fonction(paramètre1,p2):

• Appel : fonction()

• Fonction lamba

Def fonction(p1, p2):

Return p1 + p2

=

Lambda param1, param2 : param1 + param2

Appeler la fonction :

Ma\_fonction(1,2)

=

(Lambda param1, param2 : param1 + param2)(1,2)

• fonctions lamda souvent utilisées pour les fonctions map, filter, reduce

**— Map = appliquer une fonction sur chaque élément d’une liste**

Liste = [1, 2, 3, 4, 5]

Liste2 = list(map(lamda x : x \* 2, liste))

Print(liste2) # [2, 4, …]

**— Filter = filtrer ne liste en fonction d’une condition**

Liste2 = list(filter(lambda x : x % 2 == 0, liste))

Print(liste2) # [2, 4]

**— Reduce = réduire une liste en ne seule valeur**

From functools import reduce

Resultat = reduce(lamda x,y : x + y, liste)

Print(resultat) #15

Héritage, polymorphisme, encapsulation