

**CHAPITRE 1: LES NOTIONS DE BASE DU LANGAGE PYTHON**

## Amara Dalila

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

[dalilaa.amara@gmail.com](mailto:dalilaa.amara@gmail.com)



**PARTIE I: VOCABULAIRE DE BASE ET LES**

**Structures de Contrôle**

## Amara Dalila

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

[dalilaa.amara@gmail.com](mailto:dalilaa.amara@gmail.com)

# Python: Pourquoi?

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Python est un langage de programmation populaire. Il a été créé par *Guido van Rossum et sorti en 1991.* Il permet de:



Créer des applications Web.

Se connecter aux bases de données.

Manipuler des fichiers de divers types (ouverture et modification)

Il gère les données de volume important avec un traitement des mathématiques complexes

Permet de développer des applications de production, Il assure l’analyse des données,

# Python: Compilé ou interprété?

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Un programme écrit par un logiciel est appelé code source.



•

•

•

La traduction d’un programme source en code binaire exécutable par

la machine utilise les techniques : l’interprétation ou la compilation

**Interprétation:** chaque ligne du programme source analysé est traduite au fur et à mesure en quelques instructions du langage machine, qui sont ensuite directement exécutées. Aucun programme objet n’est généré.



# Python: Compilé ou interprété?

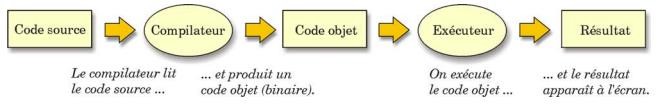
Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

**Compilation:** traduit la totalité du code source en une fois pour produire une nouvelle suite de codes que l’on appelle programme objet (ou code objet). Celui-ci peut désormais être exécuté indépendamment du compilateur et être conservé tel quel dans un fichier (« fichier exécutable »).



•



**Source:** (Swinnen, 2009)

# Python: Compilé ou interprété?

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

* L’interprétation est idéale lorsque l’on est en phase d’apprentissage du langage, elle teste au fur et à mesure toute modification apportée au code source, sans passer par une phase de compilation qui demande toujours du temps,
* La compilation est préférable Lorsqu’un projet comporte des fonctionnalités complexes qui doivent s’exécuter rapidement, il est clair en effet qu’un programme compilé fonctionnera toujours nettement plus vite que son homologue interprété, puisque dans cette technique l’ordinateur n’a plus à (re)traduire chaque instruction en code binaire avant qu’elle puisse être exécutée.

# Python: Compilé ou interprété?

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Python est un langage moderne qui combine les 2 techniques : interprété, compilé,



Lorsque vous lui fournissez un programme source, Python commence par le compiler pour produire un code intermédiaire, similaire à un langage machine, que l’on appelle bytecode, lequel sera ensuite transmis à un interpréteur pour l’exécution finale. Du point de vue de l’ordinateur, le bytecode est très facile à interpréter en langage machine. Cette interprétation sera donc beaucoup plus rapide que celle d’un code source.

# Python: les points forts

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

 Python fonctionne sur différentes plateformes (Windows, Mac, Linux, Raspberry Pi, etc.).

 Il utilise une syntaxe simple proche de la langue anglaise.

 Python permet d'écrire des programmes avec moins de lignes

 Python peut être traité de manière procédurale, orientée objet ou fonctionnelle.



* Il existe 2 versions de Python: 2.x et 3.x, dans ce cours on va utiliser la version 3.7.
* Il est possible d'écrire Python avec un éditeur ou en utilisant un

environnement de développement (voir TP pour l’installation)

# Structure d’un programme python?

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

**Importation des**

**modules**

**Commentaire sur une ligne**

**Déclaration des**

**variables**

**Commentaire sur**

**plusieurs lignes**



**Rédaction des instructions à l’aide des fonctions et des boucles**

Introduction

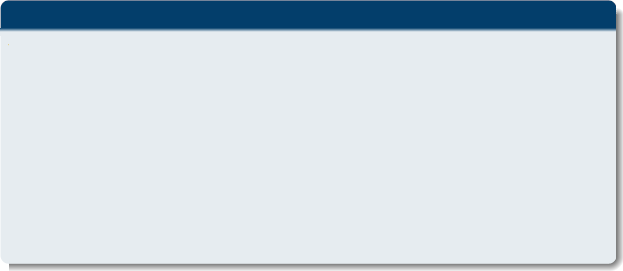
Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les variables

Utilisation des variables



Python n'a pas de commande pour déclarer une variable.

Une variable est créée au moment où vous lui attribuez une valeur pour

la première fois.

Une variable dans Python est constituée de deux éléments : son nom et sa valeur: **Nomvariable= valeur**

Pas besoin de déclarer le type de la variable. Son type change en changeant sa valeur



**NB.** Sous Python, on peut assigner une valeur à plusieurs variables simultanément. Exemple : x = y = 7

On peut aussi effectuer des affectations parallèles à l’aide d’un seul

opérateur : a, b = 4, 8.33

Introduction

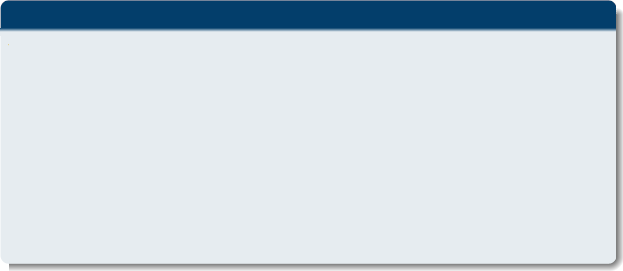
Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les variables

Contraintes sur les noms des variables



Un nom de variable est une séquence de lettres (a → z , A → Z) et de chiffres (0 → 9), qui doit toujours commencer par une lettre.

Seules les lettres ordinaires sont autorisées.

Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que $, #, @, etc. sont interdits, à l’exception du caractère \_ (souligné). La casse est significative (les caractères majuscules et minuscules sont distingués). **NB** : Ahmed, ahmed, AHMED sont donc des variables différentes.

Les mots réservés au langage ne peuvent pas être utilisés comme noms de variables (del, and, if, elif, try, while, for, break, class, return, print, etc)



Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les types de données



|  |  |  |
| --- | --- | --- |
| **Type chaine de caractères** | | |
| **str** | Chaines de caractères | Ch=‘bievenue’ type(ch) # <class 'str'> |
| **Types numériques** | | |
| **int** | Entiers | X=5  type(X) <class 'int'> |
| **float** | Nombres à virgule flottante | X=5.0  type(X) <class ‘float'> |
| **Type booléen** | | |
| **bool** | Valeur True ou False | X=False  type(X) <class ‘bool'> |

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les types de données

|  |  |  |
| --- | --- | --- |
| **Type séquences** | | |
| **list** | Collection d’éléments | jour = ["lundi", "mardi", "mercredi", 1800, 20.357, "jeudi", "vendredi"] type(jour) #<class 'list'> |
| **tuple** | Collection d’éléments | jour = ("lundi", "mardi", "mercredi", 1800, 20.357, "jeudi", "vendredi« ) type(jour) #<class ‘tuple'> |

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: les entrées/sorties



Affichage des données

L'équivalent de la fonction Ecrire en *python* est la fonction **print().**

Il faut lui passer la liste des données que l'on veut afficher (séparées

par des virgules).

### Exemple:

res = 10/3

print("le resultat est ",res,"\n")

**NB.** \n fait un retour à la ligne

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: les entrées/sorties

La saisie des données



L' équivalent de la fonction Lire en Python est la fonction **input().**

Il est possible de rajouter un message en argument, qui sera affiché juste avant d'effectuer la saisie.

Le résultat renvoyé par input est de type chaîne de caractère (str) : pour le convertir en entier ou flottant il faut respectivement utiliser les fonctions int() ou float().

### Exemples:

saisie = int(input("Veuillez saisir un entier"))

nom=input()

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les opérateurs arithmétiques

|  |  |  |
| --- | --- | --- |
| **Opérateur** | **Nom** | **Exemple** |
| + | addition | x+y |
| - | soustraction | x-y |
| \* | multiplication | x\*y |
| / | division | x/y |
| % | reste de la division Euclidienne % | x%y |
| \*\* | puissance | x\*\*y |
| // | quotient de la division Euclidienne | x//y |

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les opérateurs d’affectation



|  |  |
| --- | --- |
| **Opérateur** | **Exemple** |
| = | x=2 |
| += | x=x+2 |
| -= | x=x-2 |
| \*= | x=x\*2 |
| /= | x=x/2 |
| %= | x=x%2 |
| //= | x=x//2 |
| \*\*= | x=x\*\*2 |
| &= | x=2 |

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les opérateurs de comparaison

|  |  |
| --- | --- |
| **Opérateur** | **Exemple** |
| == | égale x, y=3 x==y(True) |
| != | différent |
| > | supérieur à |
| < | inférieur à |
| >= | supérieur ou égale |
| <= | inférieur ou égale |

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Les variables

Les types de données Les entrées/sorties Les opérateurs

# Les bases du langage python: Les opérateurs logiques

|  |  |  |
| --- | --- | --- |
| **Operateur** | **signification** | **Exemple** |
| & | ET logique | x& y = 0 (0000 0000) |
| | | OU logique | x | y = 14 (0000 1110) |
| ~ | NON logique | ~x = -11 (1111 0101) |
| ^ | XOR | x ^ y = 14 (0000 1110) |
| >> | Décalage binaire à  droite | x>> 2 = 2 (0000 0010) |
| << | Décalage binaire à gauche | x<< 2 = 40 (0010  1000) |

Application: Exercice 2 du TP 1

# Les structures conditionnelles: La clause IF

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Syntaxe d’utilisation



Si la condition est vraie, alors l’instruction indentée (après le « :

») est exécutée.

if condition:

traitement suite

Python s'appuie sur l'indentation (espace blanc au début d'une ligne) pour définir la portée dans le code (au lieu des accolades pour d’autres langages).

### Exemples:

a,b=20,15

if b > a:

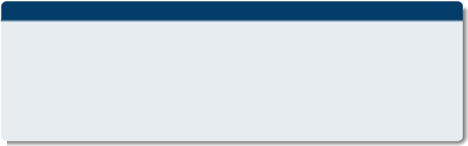
print(“b est supérieure à a") # rien ne se fasse

# Les structures conditionnelles: La clause IF..ELSE

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Syntaxe d’utilisation



l’instruction **else** permet de programmer une exécution

alternative, ou le programme doit choisir entre deux possibilités.

if condition:

traitement else:

traitement alternatif

suite

### Exemple:



a,b=20,15

if b > a:

print("b est supérieure à a") # rien ne se fasse

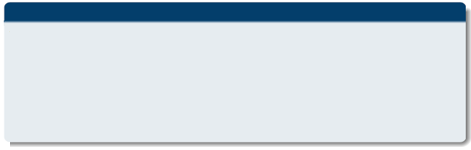
else: print("b est supérieure à a")

# Les structures conditionnelles: La clause IF..ELIF..ELSE

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Syntaxe d’utilisation



Le mot-clé **elif** est utilisé pour dire si les conditions précédentes

n'étaient pas vraies, alors essayez cette condition.

if condition:

traitement elif condition1:

traitement alternatif1 else: traitement alternatif2 suite

# Les structures conditionnelles: La clause IF..ELIF..ELSE

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

### Exemple:

a = 25

b = 14

if b > a:

print("b supérieure à a")

elif a == b:

print("a et b sont égaux")

else:

print("a est supérieur à b")



# Les structures itératives

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

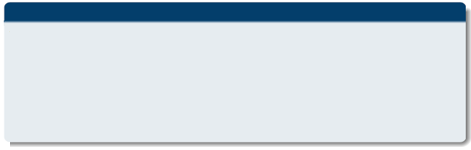


Python utilise deux répétitions:

* La boucle For
* La boucle While

La boucle WHILE

Syntaxe d’utilisation



La boucle **while** permet d’exécuter un ensemble d'instructions tant qu'une condition est vraie.

Il est possible d’arrêter la boucle même si la condition while est vraie

avec l'instruction **break,**

Il est aussi possible d’arrêter l'itération en cours et continuer avec la suivante avec l’instruction **continue:,**



# Les structures itératives La boucle WHILE

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives



**Exemple:**

i = 0

while i < 6:

i += 1

if i == 3:

continue

print(i)

#Le nombre 3

est manquant



**Exemple:**

i = 1

while i < 6: print(i) if i == 3:

break

i += 1

**Exemple:**

i = 1

while i < 6: print(i) i += 1

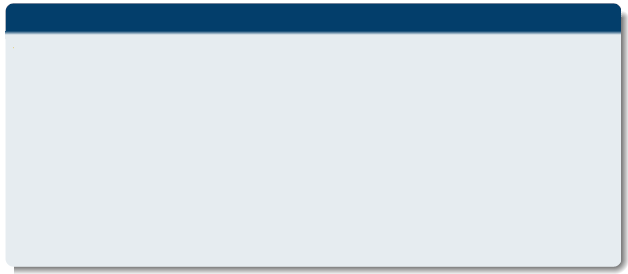
#Ce bout de code permet d’afficher i tant que i est inférieur à 6

Les structures itératives La boucle FOR

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

Syntaxe d’utilisation



La boucle **for** permet de parcourir une séquence

for i in sequence:

instruction

Une séquence peut être une liste, un tuple, une chaine de caractères,

Il est possible d’arrêter la boucle avant même de parcourir tous les éléments avec l’instruction **break**

Il est aussi possible d’arrêter l'itération en cours et continuer avec la suivante avec l’instruction **continue:,**

Les structures itératives La boucle FOR



**Exemple:**

chaine = "a b c d"

for x in chaine: if x == "a": continue

print(x)

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives



**Exemple:**

chaine = "a b c

d"

for x in

chaine:

print(x)

if x == "a":

break

**Exemple:** chaine = "ma chaine"

for x in chaine:

print(x)

Les structures itératives

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

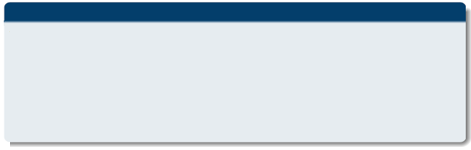
La boucle FOR: La fonction range()



Pour faire des itérations avec une boucle for un nombre de fois

spécifié, on utilise la fonction **range()**

Syntaxe d’utilisation



La fonction **range(début, fin, pas),** qui va générer une liste de nombres selon trois paramètres :

* **début :** le premier nombre de la séquence (0 par défaut);
* **fin :** correspond au dernier nombre de la séquence non inclus. La

fonction va générer des nombres de début à fin- 1;

* **pas :** le pas entre chaque nombre généré (1 par défaut).

# Les structures itératives

Introduction

Les bases du langage Python Les structures de contrôle conditionnelle Les structures de contrôle itératives

La boucle FOR: La fonction range()



**Exemple:**

for x in range(2, 30

, 3):

print(x)

**Exemple:**

for x in range

(2, 6):

print(x)

**Exemple:**

for x in range(6):

print(x)



**PARTIE II: LES COLLECTIONS EN PYTHON**

## Amara Dalila

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

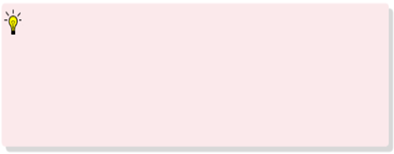
[dalilaa.amara@gmail.com](mailto:dalilaa.amara@gmail.com)



# Les collections: Pourquoi?

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

* + Le problème avec les variables c’est qu’une variable x=5



ne peut contenir qu’une seule valeur

* + On souhaite avoir des variables qui contiennent plusieurs

valeurs=une liste de valeurs= une collection

* + Deux types de collections sont utilisées en python:
    - **les séquences** ( **les listes**, **les tuples, string,set** )
    - **Les dictionnaires**

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

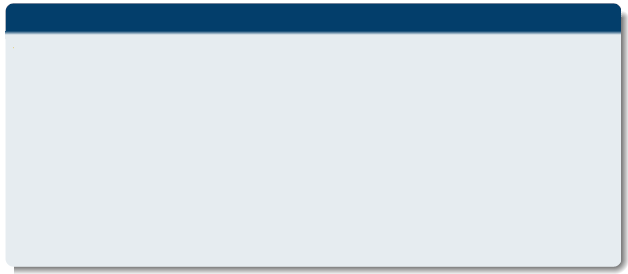
Formatage des chaines

# Les collections (séquences): listes

 Une séquence est un ensemble de données ordonnés, il y a un début et une fin.

 Chaque élément a un index de 0 à **n-1**. Ainsi on peut accéder à nos

éléments via ces index



Syntaxe d’utilisation

On utilise les [ ] pour indiquer une lise Il est possible de modifier une liste Une liste autorise les doublons

Il est possible d’utiliser le constructeur **list()** pour créer une liste

**Exemples:**

liste\_1=[1,4,4,2,7,35,84]

villes=['paris', 'berlin','londres', 'bruelles'] liste\_2=[liste\_1, villes]

liste\_3=[]

ListeEtrange = [4, 10.2, True, "Georges Dupond", ["une autre liste", 1]] maliste = list(("apple", "banana", "cherry")) # double parenthèse print(maliste)

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

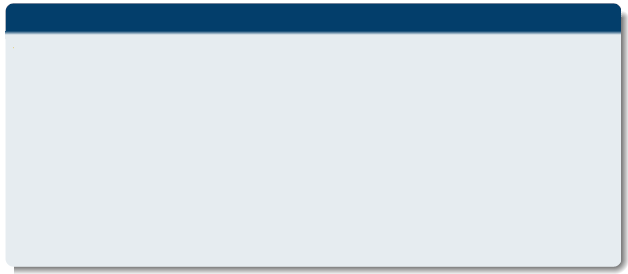
Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections (séquences): tuples

Syntaxe d’utilisation



On utilise les ( ) pour indiquer les tuples

Un tuple est comme une liste :ordonné et indexé, mais une fois crées on

ne peut pas le modifier, il est donc protégé,

Les éléments d’un tuple peuvent être de différents types

Un tuple est plus rapide en temps d’exécution par rapport à une liste.

*En général on utilise les listes*

**Exemples:**

tuple\_1=(5,7,7,8,10, True, ‘Femme’)

print(tuple\_1) *# (5, 7, 7,8, 10, True, ‘Femme’)*

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections(séquences): Accès aux éléments



* Il existe 2 techniques pour accéder aux éléments d’une séquence :
  + **indexing et**
  + **slicing**

**Indexing** : Cette technique permet d’accéder directement à un élément par

son index commençant à partir de 0

**Exemples:**

villes=[‘Tunisie', 'berlin','londres', ‘bruxelles’] montuple = ("pomme", "banane", "cerise") machaine= "je suis un étudiant en 2DSI” print(villes[0]) *#premier élément Tunisie* print(villes[1]) *# second élément berlin* print(villes[3]) *#dernier élément bruxelles*



print(montuple[1]) *#second élément du tuple banane*

Print(machaine[1]) *#second élément de machaine*

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections (séquences): Accès aux éléments



* NB. Les **index négatifs** indiquent qu’on commence à partir de la

fin d’une séquence

**Exemples:**

villes=[‘Tunisie', 'berlin','londres', ‘bruxelles’]

montuple = (“pomme", "banane", "cerise") machaine= “la maman est une perle”

print(villes[-1])*#dernier élément bruxelles* print(villes[-2])*#avant dernier élément londres* print(motuple[-1]) *#cerise*



print(machaine[-1]) *#e*

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections (séquences): Accès aux éléments

**Slicing** : Le slicing permet d’accéder à une fourchette d’éléments d’une séquence en indiquant une fourchette d’index : **index début**, **index fin, pas :**

print(villes[0:3]) *# [‘Tunisie', 'berlin', 'londres'], on peut faire* print(villes[:3]) *car par défaut on commence par l’index 0* print(villes[-1]) *# afficher le dernier élément*

print(villes[2 :]) *#affiche à partir de l’index 2 jusqu’à le dernier élément*

print(villes[1 :3 :2]) *#par défaut le pas est 1, # ['berlin']*

print(villes[ : :2]) *# [‘Tunisie', 'londres’]*

print(villes[ : :-1]) *# extraire tous les éléments de la liste mais à l’envers : [‘bruxelles', 'londres', 'berlin', ‘Tunisie’] de même pour les chaines et les tuples ou les chaine de caractères:*

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections (séquences): Modification



* NB. Il est possible de **modifier** les valeurs d’une séquence en

utilisant l’opérateur d’affectation **=**

**Exemples:**

villes=[‘Tunisie', 'berlin','londres', ‘bruxelles’]

montuple = ("pomme", "banane", "cerise")

machaine= "la maman est une perle”

villes[0]=5 print(villes[0]) *#5* montuple[1]= “tomate “ print(montuple[1]) *#tomate* machaine[0]=m Print(machaine[0]) *#m*



Application: Exercice 1 du TP 2

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les listes



* NB. Il existe un ensemble de méthodes prédéfinies sur les

séquences

* Il suffit de taper le nom de la séquence (liste, tuple, chaine) suivi par un point pour voir la liste des méthodes.

|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** | **Exemples** |
| **append(element)** | Rajoute un élément a la fin de celle-ci. | villes.append('Gabes’) print(villes) # [‘Tunisie', 'berlin', 'londres', 'bruelles', 'Gabes'] |
| **insert(index,**  **element)** | Rajoute un  élément a  l’index donné | villes.insert(2, ‘madrid’) print(villes) # ['paris', 'berlin', 'madrid', 'londres', 'bruelles', 'Gabes'] |
| **copy** | Copie une liste | Ville1=villes.copy() |

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les listes

|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** | **Exemples** |
| **sort()** | trier une liste par défaut en ordre alphabétique | villes=['paris', 'berlin', 'madrid', 'londres', 'bruelles', 'Gabes', 'Rome', 'Quatar’]  print(villes.sort())# ['berlin',  'bruelles', 'gabes', 'londres',  'madrid', 'paris', 'quatar', 'rome’] print(villes.sort(reverse=True)) # ['rome', 'quatar', 'paris', 'madrid', 'londres', 'gabes', 'bruelles', 'berlin'] |
| **clear()** | Vide une liste | print(villes.clear()) # [] |
| **len()** | Longueur  d’une liste | print(villes.len()) # 8 |

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les listes

|  |  |
| --- | --- |
| **Opération** | **Description** |
| **count(valeur) :** | Compter combien de fois une valeur apparait dans ma  liste |
| **remove(), pop() del()** | **remove** supprime la première occurrence d’un élément  **pop** supprime le dernier élément  **del** supprime un élément par son indice |
| **index()** | trouver l’indice de la première occurrence d’un élément  dans la liste |
| **+** | Joindre 2 listes |

Application: Exercice 2, 3 du TP 2



Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les tuples



|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** |  |
| **count(valeur)**  **:** | Compter combien de fois une valeur apparait dans mon | montuple =  (1, 3, 7, 8, 7, 5, 4, 6, 8,  5) |
|  | tuple | x = montuple.count(5) |
|  |  | print(x) #2 |
| **index()** | cherche l’indice de la première occurrence | montuple =  (1, 3, 7, 8, 7, 5, 4, 6, 8, |
|  | d’un élément dans un | 5) |
|  | tuple | x = montuple.index(8) |
|  |  | print(x) #3 |
| **+**  Application: Exerc | | |

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les chaines

|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** |  |
| **len()** | Longueur d’une chaine |  |
| **index()** | cherche l’indice de la première occurrence d’un élément dans une chaine | txt = "Hello, welcome to my world."  x = txt.index("welcome")  print(x) # 7 |
| **+** | Concaténer 2 chaines | a = "Hello"  b = "World"  c = a + b  print(c) # “Hello World” |

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Les opérations sur les chaines

|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** |  |
| **len()** | Longueur d’une chaine |  |
| **index()** | cherche l’indice de la première occurrence d’un élément dans une chaine | txt = "Hello, welcome to my world."  x = txt.index("welcome")  print(x) # 7 |
| **+** | Concaténer 2 chaines | a = "Hello"  b = "World"  c = a + b  print(c) # “Hello World” |

### Liste des méthodes sur les chaines



<https://www.w3schools.com/python/python_strings_methods.asp>

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

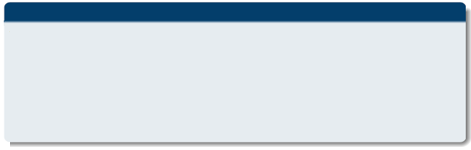
Formatage des chaines

# Les collections-Les séquences: Formatage des chaines



* Il est possible d’afficher une chaine (ou nombres) comme il est souhaité en utilisant la fonction **format(),**
* Il suffit d’ajouter des accolades **{ }** dans le texte et exécutez les

valeurs via la méthode format() :



Syntaxe d’utilisation

*string*.format(*value1, value2...*)

**Exemples:**

prix = 44

txt = "Le prix est {} dinars"

print(txt.format(prix))

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Les listes

Les tuples

Opérations sur les séquences

Formatage des chaines

# Les collections-Les séquences: Formatage des chaines



* À l'intérieur des **{ },** on peut ajouter un type de formatage pour

formater le résultat :

|  |  |  |
| --- | --- | --- |
| **Type** |  |  |
| :f | Aligne le  résultat à gauche (droite) selon la valeur donnée | txt = "le prix est {:.2f} dinars."  print(txt.format(45))  txt = "le prix est {:f} dinars."  print(txt.format(45)) |
| :b  (:o octal, :d  décimal, etc) | Équivalent  binaire | txt = "L'équivalent binaire de {0}  est {0:b}"  print(txt.format(5)) |

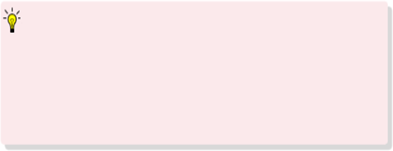
### Liste des types de formatage



<https://www.w3schools.com/python/ref_string_format.asp>

# Les collections-Les dictionnaires:

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

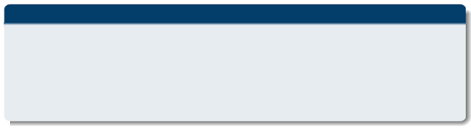


* Les séquences sont des structures de données ou les valeurs ont un

ordre selon un index.

* Les dictionnaires sont aussi des structures de données qui contrairement aux séquences ne sont pas ordonnées (avant la version 3,7) : les valeurs n’ont pas d’index mais des **cléfs.**

Syntaxe d’utilisation



Un dictionnaire est un ensemble d’affectations clef : valeur

**NB.** Le nom de la clé doit être entre guillemet ou entre apostrophe

**NB.** Une clé doit être unique : on ne peut pas avoir deux clefs de même

nom dans un même dictionnaire

# Les collections-Les dictionnaires: Accès, modification, Ajout

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

**Exemples:**

traduction={

"chien":"dog",

"chat":"cat",

"souris":"mouse",

"oiseau":"bird" }

print(traduction) *# {'chien': 'dog', 'chat': 'cat', 'souris':*

*'mouse', 'oiseau': 'bird’}* print(traduction["oiseau"]) *#bird* traduction["chien"]= "dogs"

print(traduction) *# {'chien': ‘dogs', 'chat': 'cat', 'souris':*

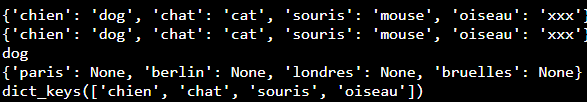
*'mouse', 'oiseau': 'bird’}*

traduction["papillon"]= "butterfly"



print(traduction) *# {'chien': ‘dogs', 'chat': 'cat', 'souris':*

*'mouse', 'oiseau': 'bird’, ‘papillon': ‘butterfly’}*





# Les collections-Les dictionnaires: Les méthodes

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

|  |  |  |
| --- | --- | --- |
| **Opération** | **Description** | **Exemples** |
| **clear()** | Supprime tous les éléments du dictionnaire |  |
| **copy()** | Renvoie un dictionnaire avec les clés et la valeur spécifiées |  |
| **fromkeys()** | fromkeys() |  |
| **get()** | get() Renvoie la valeur de la clé spécifiée |  |
| **Keys()** | Retourne une liste contenant les clés du dictionnaire |  |

Les collections-Les dictionnaires: Les méthodes

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

|  |  |
| --- | --- |
| **Opération** | **Description** |
| **pop()** | pop() Supprime l'élément avec la clé spécifiée |
| **values()** | Renvoie une liste de toutes les valeurs du dictionnaire |

<https://www.w3schools.com/python/python_dictionaries_methods.asp>

# Les collections-Les dictionnaires: Les méthodes



* **NB.** On peut nester (Nested) un dictionnaire : c’est-à-dire créer un

dictionnaire qui contient d’autres dictionnaires

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

### Exemples :

inventaire={

"bananes":2000, "figues":3000, "pommes":5000

}

dictionnaire\_3={ "dict\_1":traduction, "dict\_2":inventaire



}

print(dictionnaire\_3) *# {'dict\_1': {'chien': 'dog', 'chat': 'cat',*

*'souris': 'mouse', 'oiseau': 'bird'}, 'dict\_2': {'bananes': 2000,*

*'figues': 3000, 'pommes': 5000}}*

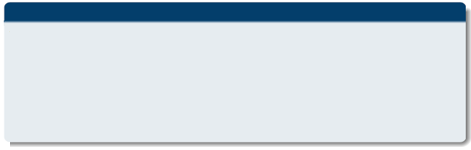
# Parcours des collections: Les listes



* **NB.** On peut parcourir les valeurs d’une liste par la boucle **for**

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

Syntaxe d’utilisation



for *i in liste*

### Exemples:

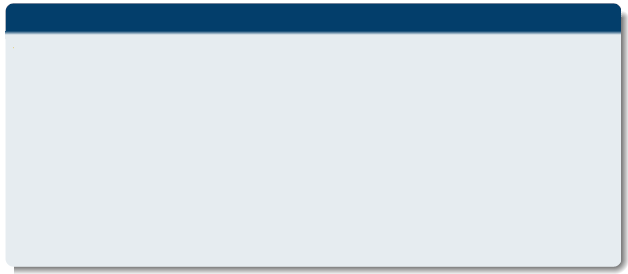
maliste =

["pomme", "banane", "cerise"]

for i in maliste:

print(maliste[i])

# Parcours des collections: Les listes



Les collections: les séquences Les collections: les dictionnaires Parcours des collections

* **NB.** Il est possible de parcourir les éléments de la liste par leur

numéro d'index en utilisant les fonctions **range()** et **len()**

Syntaxe d’utilisation

format *i in range(len(liste))*

### Exemples:

maliste =

["pomme", "banane", "cerise"] for i in range(len(maliste)): #[0, 1, 2]

print(maliste[i])

thislist =

["apple", "banana", "cherry"]

i = 0

while i < len(thislist):

print(thislist[i])

i = i + 1

* + **NB.** Le parcours des tuples se fait de la même façon que les listes

# Parcours des collections: Les listes



* **NB.** Il est possible de créer une nouvelle liste basée sur les valeurs d'une liste existante en utilisant une syntaxe courte: liste en compréhension (**list comprehension**)

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

### Exemples :

Créer une nouvelle liste à partir de la liste des fruits, contenant

uniquement les fruits avec la lettre "a" dans le nom.

fruits =

["pomme", "banane", "cerise", "k

iwi", "manga"] newlist = []

for x in fruits:

if "a" in x: newlist.append(x)

print(newlist)



fruits =

["apple", "banana", "cherry"

, "kiwi", "mango"]

newlist =

[x for x in fruits if "a" in

x]

print(newlist)



Parcours des collections: Les dictionnaires

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Syntaxe** | | | **Exemple** |
|  | for x  print(x) | in | dict: |  |
|  |  | | | dict = { |
| **Parcours des cléfs** | "Marque": "Ford",  "modèle": "Mustang",  "année": 1964 |
|  | } |
|  | for x in dict.keys(): print(x) | | |
| for x in dict: |
| print(x) |
| # ou bien |
| for x in dict.keys(): |
| print(x) |



Parcours des collections: Les dictionnaires

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

|  |  |  |
| --- | --- | --- |
| **Type** | **Syntaxe** | **Exemple** |
|  | for x in dict:  print(dict[x]) |  |
| **Parcours des valeurs** |  | dict = {  "Marque": "Ford",  "modèle": "Mustang", "année": 1964  }  for x in dict:  print(dict[x])  # ou bien  for x in dict.values():  print(x) |
|  | for x in dict. values():  print(x) |

Parcours des collections: Les dictionnaires

Les collections: les séquences Les collections: les dictionnaires Parcours des collections

|  |  |  |
| --- | --- | --- |
| **Type** | **Syntaxe** | **Exemple** |
| Parcours des cléfs et des valeurs | for x,  y in dict.items(): print(x, y) | dict = {  "Marque": "Ford",  "modèle": "Mustang",  "année": 1964  }  for x,y in dict.items():  print(x,y) |

