



HAL
open science

Cours d'introduction Python - Stéphane Guinard

Stéphane Guinard

► **To cite this version:**

Stéphane Guinard. Cours d'introduction Python - Stéphane Guinard. École d'ingénieur. France. 2018. cel-02182392

HAL Id: cel-02182392

<https://hal.science/cel-02182392>

Submitted on 12 Jul 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Cours d'introduction Python

ENSG - IT1

Stéphane Guinard

v1.1.0

19/10/2018



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Jul 27 17:18:17 2017
4
5 gauthor: Stephane Guinard
6 """
7
8 import tirage_points
9
10 import numpy as np
11
12 def ts_inside_cercle(x_point,y_point,x_cercle=0,y_cercle=0,r_cercle=1):
13     """
14     Fonction qui teste l'appartenance d'un point (x_point, y_point)
15     à un cercle de centre (x_cercle, y_cercle) et de rayon r_cercle
16
17     :param x_point: abscisse du point dont on cherche à connaître l'appartenance ou non au cercle
18     :param y_point: ordonnée du point
19     :param x_cercle: abscisse du centre du cercle, par défaut 0
20     :param y_cercle: ordonnée du centre du cercle, par défaut 0
21     :param r_cercle: rayon du cercle, par défaut 1
22     :return: True/False suivant que le point appartienne au cercle
23     """
24     d = np.sqrt((x_point-x_cercle)*(x_point-x_cercle) + (y_point-y_cercle)*(y_point-y_cercle))
25     return True if d<1 else False
26
27 def nb_points_sub_domain(points):
28     """
29     Fonction qui calcule le nombre de points appartenant à un sous-domaine particulier
30     pour une liste de points donnée
31
32     :param points: liste de points
33     :return: nombre de points appartenant à un sous-domaine
34     """
35     nb_points = 0
36     # On parcourt la liste de points et on cherche son appartenance au cercle trigonométrique
37     for point in points:
38         if ts_inside_cercle(point[0],point[1]) == True:
39             nb_points += 1
40     print (nb_points)
41     return nb_points
42
43
44 # Tests unitaires
45 if __name__ == "__main__":
46     # obtenir une liste de points
47     points = tirage_points.find_points(10)
48     # tester si un point appartient à un cercle
49     print(ts_inside_cercle(0.5,0.5)) # True
50     print(ts_inside_cercle(0.9,0.9)) # False
51     # Compter le nombre de points dans un sous-domaine
52     n = nb_points_sub_domain(points)
```



Sommaire

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

6 Fichiers et itérateurs

7 Conclusion



Déroulement de la présentation

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- 1 **Présentation du module**
- 2 Présentation de Python
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Objectifs

► Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Comprendre les bases de Python et son intérêt



Objectifs

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- Comprendre les bases de Python et son intérêt
- Savoir manipuler différents types de données



Objectifs

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- Comprendre les bases de Python et son intérêt
- Savoir manipuler différents types de données
- Savoir créer un programme complet en Python



Objectifs

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- Comprendre les bases de Python et son intérêt
- Savoir manipuler différents types de données
- Savoir créer un programme complet en Python
- Prendre des bonnes habitudes de programmation (commentaires, documentation, organisation du code, ...)



Objectifs

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- Comprendre les bases de Python et son intérêt
 - Savoir manipuler différents types de données
 - Savoir créer un programme complet en Python
 - Prendre des bonnes habitudes de programmation (commentaires, documentation, organisation du code, ...)
- ⇒ Maîtriser un premier langage informatique.



Séances

► Présentation du module

7 séances

Présentation de Python

- 19/10 aprem : cours théorique

Types et Opérations

- 24/10 aprem : TP

Syntaxe

- 25/10 matin : TP

Modules et fonctions

- 26/10 aprem : cours + TP

Fichiers et itérateurs

- 06/11 aprem : TP

Conclusion

- 13/11 aprem : TP

References

- 16/11 aprem : TP
- 28/11 matin : évaluation



Organisation

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

3 groupes avec chacun 1 encadrant.

Encadrement :

- Mohamed Boussaha
- Victor Coindet
- Stéphane Guinard



Évaluation

► Présentation du module

Présentation de Python

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

TP noté (29/11 matin)

- sur machines
- individuel
- accès au cours





Déroulement de la présentation

Présentation du module

► Présentation de Python

Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Déroulement de la présentation

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 **Présentation de Python**
 - Historique
 - Caractéristiques du langage
 - Exécution
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Historique

Présentation
du module

Présentation
de Python

- Historique
- Caractéristiques
du langage
- Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 :
développement d'ABC

CWI



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0

CWI



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2

CWI



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2
- 1999 : version 1.6

CWI

CNRI



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2
- 1999 : version 1.6
- 2000 : version 2.0



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2
- 1999 : version 1.6
- 2000 : version 2.0
- 2008 : version 3.0



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2
- 1999 : version 1.6
- 2000 : version 2.0
- 2008 : version 3.0
- 2010 : version 2.7



Guido van Rossum



Historique

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Milieu des années 80 : développement d'ABC
- Février 1991 : version 0.9.0
- 1995 : version 1.2
- 1999 : version 1.6
- 2000 : version 2.0
- 2008 : version 3.0
- 2010 : version 2.7
- 2015 : **version 3.5**
- Aujourd'hui : versions 2.7.15 et 3.7.0



Guido van Rossum



License

Présentation
du module

Présentation
de Python

► Historique
Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



LGPL (Lesser GNU Public Licence)

Les outils Python peuvent être utilisés sans restriction pour produire des logiciels de tous types, même si ceux-ci sont distribués avec une licence plus restrictive que la GPL.



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Historique
▶ Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 **Présentation de Python**
 - Historique
 - Caractéristiques du langage
 - Exécution
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Caractéristiques

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python est :

- portable
- orienté objet et fonctionnel
- à typage dynamique fort
- performant
- associable
- facile à lire et à écrire
- haut niveau
- a une communauté active





Portabilité

Présentation
du module

Présentation
de Python

Historique
▶ Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Langage interprété

Langage qui n'a pas besoin d'être compilé, et peut être directement exécuté sur une machine compatible.

Leurs performances sont souvent moins bonnes que celles des langages compilés.



Portabilité

Présentation
du module

Présentation
de Python

Historique
▶ Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Langage interprété

Langage qui n'a pas besoin d'être compilé, et peut être directement exécuté sur une machine compatible.

Leurs performances sont souvent moins bonnes que celles des langages compilés.

⇒ Python est un langage interprété compatible avec les OS les plus utilisés (MS Windows, Mac OS, GNU Linux, Android, ...).



Orienté objet et fonctionnel

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Orienté Objet

Il est possible de créer des objets et d'appliquer des concepts avancés comme le polymorphisme ou l'héritage.



Orienté objet et fonctionnel

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Orienté Objet

Il est possible de créer des objets et d'appliquer des concepts avancés comme le polymorphisme ou l'héritage.

Fonctionnel

Python perçoit les instructions comme un ensemble de fonctions mathématiques à exécuter.



Typage dynamique fort

Présentation
du module

Présentation
de Python

Historique

► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il n'est pas nécessaire de déclarer le type des données mais on ne peut pas tout mélanger non plus !



Typage dynamique fort

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il n'est pas nécessaire de déclarer le type des données mais on ne peut pas tout mélanger non plus !

Python

```
toto = 5 # OK
```

```
tata = "abcde" # OK
```

```
toto + tata # NON
```

C++ / Java

```
toto = 5; // NON
```

```
int toto = 5; // OK
```

```
string tata = "abcde"; // OK
```

```
toto + tata; // NON
```



Performant

Présentation
du module

Présentation
de Python

Historique

► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

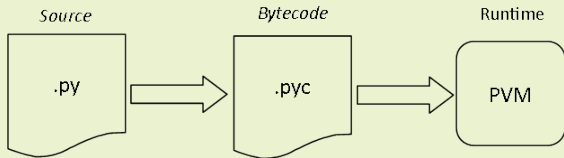
Fichiers et
itérateurs

Conclusion

References

Exécution

A l'exécution le code Python est converti en Bytecode Python, plus proche de la machine.





Performant

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

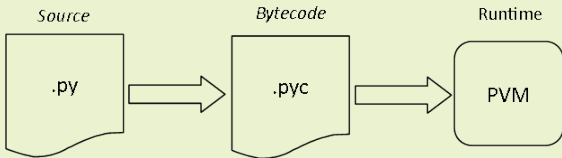
Fichiers et
itérateurs

Conclusion

References

Exécution

A l'exécution le code Python est converti en Bytecode Python, plus proche de la machine.



Garbage collector

Libère de l'espace mémoire pendant l'exécution afin de l'optimiser.



Associable

Présentation du module

Présentation de Python

- Historique
 - Caractéristiques du langage
- Exécution

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

Python peut facilement être combiné à d'autres langages :

- C/C++ : Cython
- Java : Jython
- C# : IronPython
- Perl : PyPerl
- PHP : PiP
- R : RPy





Facile à lire et à écrire

Pas de point-virgule ni d'accolades. Seule l'indentation définit la structure du code.

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References





Facile à lire et à écrire

Pas de point-virgule ni d'accolades. Seule l'indentation définit la structure du code.

⇒ Cela permet de reconnaître en un coup d'œil les principales structures d'un code.



Présentation
du module

Présentation
de Python

Historique
▶ Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Facile à lire et à écrire

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Pas de point-virgule ni d'accolades. Seule l'indentation définit la structure du code.

⇒ Cela permet de reconnaître en un coup d'œil les principales structures d'un code.

Python est reconnu comme l'un des meilleurs langages pour apprendre à programmer.





Haut niveau

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pas besoin de se soucier du matériel utilisé.



Haut niveau

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pas besoin de se soucier du matériel utilisé.
- Pas besoin de gérer la mémoire.



Haut niveau

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage

Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pas besoin de se soucier du matériel utilisé.
- Pas besoin de gérer la mémoire.
- Proche du langage naturel.



Haut niveau

Présentation du module

Présentation de Python

Historique

- ▶ Caractéristiques
du langage

Exécution

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- Pas besoin de se soucier du matériel utilisé.
- Pas besoin de gérer la mémoire.
- Proche du langage naturel.
- Peut être moins efficace que des langages bas-niveau.



Communauté

Présentation
du module

Présentation
de Python

Historique
► Caractéristiques
du langage
Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

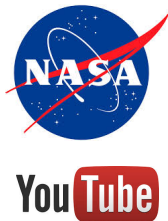
Conclusion

References

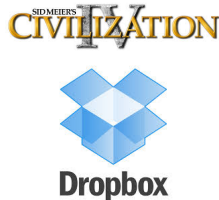
Python est très utilisé :



P X A R



YouTube



Dropbox



Communauté

Présentation du module

Présentation de Python

- Historique
- Caractéristiques du langage
- Exécution

Types et Opérations

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

Communauté active sur Internet :

- Top 10 des tags les plus populaires de StackOverflow
- Wiki : wiki.python.org/
- De nombreux forums spécialisés :
 - python-forum.io/
 - python-forum.org/
 - openclassrooms.com/forum/categorie/langage-python





Déroulement de la présentation

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 **Présentation de Python**
 - Historique
 - Caractéristiques du langage
 - Exécution
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Interpréteur Python

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Lorsque l'on exécute un code Python, un programme lit le code et exécute les instructions qu'il contient. Ce programme est appelé un interpréteur Python.



Interpréteur Python

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Lorsque l'on exécute un code Python, un programme lit le code et exécute les instructions qu'il contient. Ce programme est appelé un interpréteur Python.

Un interpréteur est systématiquement installé lorsqu'on installe Python.



Interpréteur Python

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Lorsque l'on exécute un code Python, un programme lit le code et exécute les instructions qu'il contient. Ce programme est appelé un interpréteur Python.

Un interpréteur est systématiquement installé lorsqu'on installe Python.

Attention

Plusieurs versions de Python peuvent cohabiter sur une même machine.



Exécution

À l'exécution, le code Python est converti en Bytecode Python, puis interprété par une machine virtuelle.

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

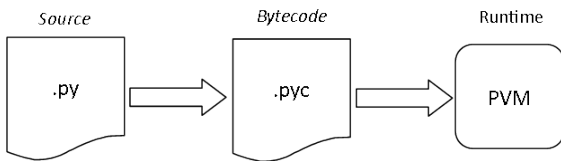
Conclusion

References



Exécution

À l'exécution, le code Python est converti en Bytecode Python, puis interprété par une machine virtuelle.



Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
▶ Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Exécution

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

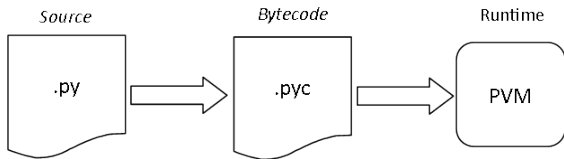
Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

À l'exécution, le code Python est converti en Bytecode Python, puis interprété par une machine virtuelle.



Vitesse d'exécution

L'exécution est plus rapide que pour un langage interprété classique car la machine virtuelle n'a pas besoin de ré-analyser et de re-parser le code. Cependant il reste plus lent que les langages compilés car la PVM doit interpréter le Bytecode.



Exécution

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Terminal

```
sguinard@RKS1306W220-Ubuntu:~$ python3 test.py  
hello world  
sguinard@RKS1306W220-Ubuntu:~$
```



Exécution

Présentation
du module

Présentation
de Python

Historique
Caractéristiques
du langage
► Exécution

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Terminal

```
sguinard@RKS1306W220-Ubuntu:~$ python3 test.py  
hello world  
sguinard@RKS1306W220-Ubuntu:~$
```

Environnements de développement

Spyder (Python 3.5)

Fichier Édition Recherche Source Exécution Débuguer Consoles Outils Affichage Aide

Éditeur - /home/sguinard/test.py Console IPython

```
1 # -*- coding: utf-8 -*-  
2  
3 Created on Tue Aug 22 16:00:07 2017  
4  
5 @author: sguinard  
6  
7  
8 print("hello world")  
9
```

In [29]:

In [30]: runfile('/home/sguinard/test.py',
wdir='/home/sguinard')
hello world

In [31]:

Console Historique Console IPython

Droits d'accès : RW Fins de ligne : LF Encodage : UTF-8 Ligne 8 Colon 21 Mémoire : 75 %

► Verbose mode



Déroulement de la présentation

Présentation
du module

Présentation
de Python

► Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 Présentation de Python
- 3 **Types et Opérations**
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Variable

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Association d'un nom et d'une valeur. Elle est stockée en binaire dans la mémoire de l'ordinateur. Le nom permet à l'ordinateur de savoir quelle donnée manipuler.



Variable

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Association d'un nom et d'une valeur. Elle est stockée en binaire dans la mémoire de l'ordinateur. Le nom permet à l'ordinateur de savoir quelle donnée manipuler.

La programmation consiste essentiellement à manipuler des données stockées en binaire dans la mémoire de l'ordinateur. Pour y accéder, on utilisera des variables.



Nom

Le nom d'une variable est composé de :

- Lettres
- Chiffres
- Blancs soulignés

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Nom

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Le nom d'une variable est composé de :

- Lettres
- Chiffres
- Blancs soulignés

Interdictions

- Pas d'espaces
- Pas de caractères spéciaux (çà)
- Le premier caractère n'est pas un chiffre
- Unicité du nom
- Respecter la casse



Conventions de nommage

Dans un même code, il vous faudra être cohérent sur la notation :

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Conventions de nommage

Dans un même code, il vous faudra être cohérent sur la notation :

- exemple :
 - noms_de_variables
 - NomsDeFonctions
- notation hongroise

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Conventions de nommage

Dans un même code, il vous faudra être cohérent sur la notation :

- exemple :
 - noms_de_variables
 - NomsDeFonctions
- notation hongroise

Les variables commençant et finissant par “__” sont traditionnellement réservées au langage.

- `__name__`
- `__str__`
- `__main__`

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Mots réservés au langage

Comme dans la plupart des langages, Python dispose de mots qui lui sont réservés : il n'est pas possible de créer une variable portant l'un de ces noms.

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Mots réservés au langage

Comme dans la plupart des langages, Python dispose de mots qui lui sont réservés : il n'est pas possible de créer une variable portant l'un de ces noms.

Présentation du module

Présentation de Python

Types et Opérations

► Types

Les types numériques

Booléens

Type "rien"

Chaînes de caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

- and
- as
- assert
- break
- class
- continue
- def
- del
- elif
- else
- except
- false
- finally
- for
- from
- global
- if
- import
- in
- is
- lambda
- none
- nonlocal
- not
- or
- pass
- raise
- return
- true
- try
- while
- with
- yield



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Stockage d'une variable

La valeur d'une variable est une référence vers une adresse mémoire, à laquelle est stockée le contenu de la variable en binaire.



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Stockage d'une variable

La valeur d'une variable est une référence vers une adresse mémoire, à laquelle est stockée le contenu de la variable en binaire.

Comment savoir ce qui est lu ?



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Stockage d'une variable

La valeur d'une variable est une référence vers une adresse mémoire, à laquelle est stockée le contenu de la variable en binaire.

Comment savoir ce qui est lu ?

Python utilise des types qui permettent d'identifier le contenu.



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Stockage d'une variable

La valeur d'une variable est une référence vers une adresse mémoire, à laquelle est stockée le contenu de la variable en binaire.

Comment savoir ce qui est lu ?

Python utilise des types qui permettent d'identifier le contenu.

Contrairement à d'autres langages, pas besoin de spécifier le type, Python le déduit.



Types

Exemple : $x = 3$

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

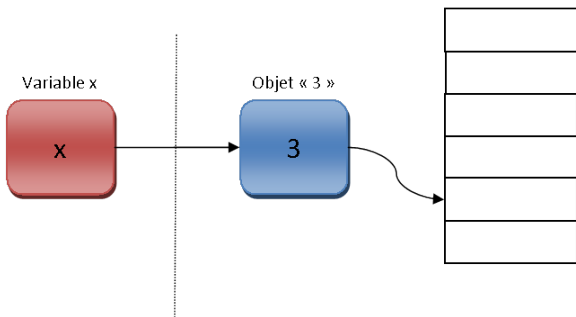


Types

Exemple : $x = 3$

- Création d'un objet représentant la valeur 3
- Création d'une variable x , si elle n'existe pas déjà
- Association de la variable x à l'objet 3

Mémoire physique



Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Types et variables

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

En Python, une variable n'est jamais associée à un type : c'est l'objet qu'elle référence qui porte le type.



Types et variables

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

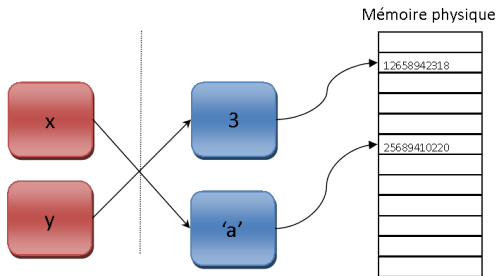
Fichiers et
itérateurs

Conclusion

References

En Python, une variable n'est jamais associée à un type : c'est l'objet qu'elle référence qui porte le type.

```
>>> x = 3
>>> y = 3
>>> x = 'a'
```





Garbage collector

Définition

Python garde un objet en mémoire tant qu'il y a une référence vers cet objet. Lorsque l'objet n'est plus référencé, un *garbage collector* est appelé pour effacer cet objet de la mémoire.

Présentation
du module

Présentation
de Python

Types et
Opérations

► Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Garbage collector

Définition

Python garde un objet en mémoire tant qu'il y a une référence vers cet objet. Lorsque l'objet n'est plus référencé, un *garbage collector* est appelé pour effacer cet objet de la mémoire.

Présentation du module

Présentation de Python

Types et Opérations

► Types

Les types numériques

Booléens

Type "rien"

Chaînes de caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

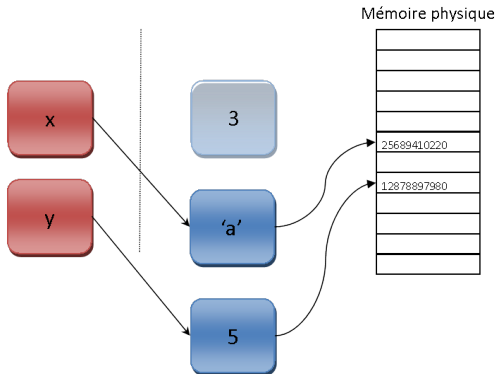
Modules et fonctions

Fichiers et itérateurs

Conclusion

References

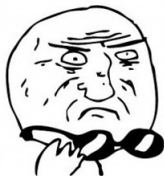
```
>>> x = 3
>>> y = 3
>>> x = 'a'
>>> y = 5
```





Types de base

Type	Exemple
Nombre	3, 1.12, -152.36941
Chaîne de caractères	"ensg", "toto titi"
Liste	[1,2,3], ["a", "b", "c"], [1, "fzi", 452.2]
Tuple	(1,2,3), ("a", "b", "c")
Collection	{1.1, "abcde", (1,2,3)}
Dictionnaire	1: "a", 2: "b"
Booléen	True, False
Rien	None



MOTHER OF GOD

Vous en voulez plus ? [▶ Verbose mode](#)

Présentation du module

Présentation de Python

Types et Opérations

▶ Types

Les types numériques

Booléens

Type "rien"

Chaînes de caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - **Les types numériques**
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Types numériques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python possède les types numériques standards de la plupart des langages, en plus de types moins courant, comme les complexes.

Type	Exemple
Integer	2,-5
Float	3.1415,-1.0
Complex	2+3j
Decimal	Decimal("0.1")
Fraction	Fraction(1,3)



Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les opérations suivantes sont valables pour les types numériques :

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
**	Puissance
/	Division flottante
//	Division entière
%	Modulo



Int et float

Il est possible de convertir simplement des nombres quelconques en “int” ou “float” avec les fonctions “int()” et “float()”.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
► Les types
numériques

Booléens
Type “rien”
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> float(a)
3.0
>>> type(a)
<class 'int'>
```




Int et float

Il est possible de convertir simplement des nombres quelconques en “int” ou “float” avec les fonctions “int()” et “float()”.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
► Les types
numériques

Booléens
Type “rien”
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> float(a)
3.0
>>> type(a)
<class 'int'>
```

Si 2 types sont mélangés dans une même expression mathématique, Python va automatiquement convertir le résultat vers le plus “complexe” des 2 types.

```
>>> type(1+3.57654)
<class 'float'>
```



Nombres complexes

Les nombres complexes sont représentés en ajoutant un “j” à la partie imaginaire.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> j*j
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'j' is not defined
>>> 1j*1j
(-1+0j)
```



Nombres complexes

Les nombres complexes sont représentés en ajoutant un “j” à la partie imaginaire.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> j*j
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'j' is not defined
>>> 1j*1j
(-1+0j)
```

Usage de “j”

Il est obligatoire d'ajouter un nombre devant le “j”, même s'il s'agit de 1 :

- j est une variable
- 1j est un nombre imaginaire



Decimal

Python possède un type “Decimal”, pour gérer les nombres décimaux avec précision.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Decimal

Python possède un type “Decimal”, pour gérer les nombres décimaux avec précision.

Float :

```
>>> 0.1 + 0.1 + 0.1 - 0.3  
5.551115123125783e-17
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Decimal

Python possède un type “Decimal”, pour gérer les nombres décimaux avec précision.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Float :

```
>>> 0.1 + 0.1 + 0.1 - 0.3  
5.551115123125783e-17
```

Decimal :

```
>>> print(Decimal('0.1') + Decimal('0.1') + Decimal('0.1')  
        ) - Decimal('0.3'))  
0.0
```



Decimal

Python possède un type “Decimal”, pour gérer les nombres décimaux avec précision.

Présentation du module

Présentation de Python

Types et Opérations

Types

► Les types numériques

Booléens

Type “rien”

Chaînes de caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et fonctions

Fichiers et itérateurs

Conclusion

References

Float :

```
>>> 0.1 + 0.1 + 0.1 - 0.3
5.551115123125783e-17
```

Decimal :

```
>>> print(Decimal('0.1') + Decimal('0.1') + Decimal('0.1')
) - Decimal('0.3'))
0.0
```

Précision Decimal

La précision du type “Decimal” n’est pas absolue non plus !

```
>>> print(Decimal(0.1))
0.1000000000000000055511151231257827021181583404541015625
```



Fraction

Python possède un dernier type numérique : “Fraction”. Il fonctionne de façon similaire au type “Decimal” et permet de gérer avec précision les fractions.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type “rien”

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Fraction

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

► Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python possède un dernier type numérique : "Fraction". Il fonctionne de façon similaire au type "Decimal" et permet de gérer avec précision les fractions.

- Sans "Fraction":

```
>>> 1/6 + 1/2  
0.6666666666666666
```

- Avec "Fraction":

```
>>> print(Fraction(1,6)+Fraction(1,2))  
2/3
```



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

► Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Booléens

Définition

Il s'agit d'objets valant soit "True" soit "False".

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

► Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Booléens

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

► Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Il s'agit d'objets valant soit "True" soit "False".

Remarque

Pendant longtemps ce type n'existait pas et on utilisait les entiers 0 pour faux et 1 pour vrai. Ceci explique que les booléens fonctionnent à peu près de la même façon que les entiers.



Booléens

Définition

Il s'agit d'objets valant soit "True" soit "False".

Remarque

Pendant longtemps ce type n'existait pas et on utilisait les entiers 0 pour faux et 1 pour vrai. Ceci explique que les booléens fonctionnent à peu près de la même façon que les entiers.

```
>>> True + 1 | >>> False == 0 | >>> False is 0  
2           | True           | False
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

► Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Booléens

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
► Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Il s'agit d'objets valant soit "True" soit "False".

Remarque

Pendant longtemps ce type n'existait pas et on utilisait les entiers 0 pour faux et 1 pour vrai. Ceci explique que les booléens fonctionnent à peu près de la même façon que les entiers.

```
>>> True + 1 | >>> False == 0 | >>> False is 0  
2           | True           | False
```

Type booléen

Les booléens ne sont pas des nombres !



Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

► Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Opérations renvoyant des booléens :

Opérateur	Signification
>	Supérieur
<	Inférieur
>=	Supérieur ou égal
<=	Inférieur ou égal
==	Égal
!=	Différent
is	Identité des objets
is not	Différence des objets



Expressions

Expressions booléennes :

Opérateur	Signification
X and Y	Vrai si X et Y valent "True"
X or Y	Vrai si X ou Y vaut "True"
not X	Vrai si X vaut "False"

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

► Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Expressions

Expressions booléennes :

Opérateur	Signification
X and Y	Vrai si X et Y valent "True"
X or Y	Vrai si X ou Y vaut "True"
not X	Vrai si X vaut "False"

<pre>if (2<3): print('toto')</pre> <p>toto</p>	<pre>if (not 2<3): print('toto')</pre>
<pre>if (2<3 and 4<3): print('toto')</pre>	<pre>if (2<3 or 4<3): print('toto')</pre> <p>toto</p>

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

► Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

► Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



None

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

► Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Il existe un type en Python qui ne peut prendre que la valeur "None". Cela signifie que la variable en question ne contient rien.



None

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

► Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Il existe un type en Python qui ne peut prendre que la valeur "None". Cela signifie que la variable en question ne contient rien.

Utilisation

On l'utilise lorsque qu'une opération n'a pas de retour, ou pour vérifier que certaines parties du programme ont bien fonctionné.



None

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
► Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Il existe un type en Python qui ne peut prendre que la valeur "None". Cela signifie que la variable en question ne contient rien.

Utilisation

On l'utilise lorsque qu'une opération n'a pas de retour, ou pour vérifier que certaines parties du programme ont bien fonctionné.

```
def test(x = None):
    if (x == None):
        print("x not specified")
    else:
        print(x)

>>> test()
x not specified

>>> test(3)
3
```



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - ▶ Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
▶ Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

En Python, il n'existe qu'un seul type de données contenant du texte : les chaînes de caractères.



Type

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

En Python, il n'existe qu'un seul type de données contenant du texte : les chaînes de caractères.

Une chaîne de caractères est définie comme une suite finie de caractères (lettres, chiffres, espaces, ...) entre guillemets.



Guillemets

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Une chaîne de caractères s'écrit entre guillemets. On peut utiliser indifféremment les simples ou doubles guillemets.



Guillemets

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Une chaîne de caractères s'écrit entre guillemets. On peut utiliser indifféremment les simples ou doubles guillemets.

```
>>> "toto" == 'toto'  
True
```



Guillemets

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Une chaîne de caractères s'écrit entre guillemets. On peut utiliser indifféremment les simples ou doubles guillemets.

```
>>> "toto" == 'toto'  
True
```

L'intérêt est de pouvoir introduire des apostrophes ou des guillemets à l'intérieur de la chaîne de caractères.



Guillemets

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Une chaîne de caractères s'écrit entre guillemets. On peut utiliser indifféremment les simples ou doubles guillemets.

```
>>> "toto" == 'toto'  
True
```

L'intérêt est de pouvoir introduire des apostrophes ou des guillemets à l'intérieur de la chaîne de caractères.

```
>>> "l'arc-en-ciel"    >>> 'le type "rien"  
"l'arc-en-ciel"      'le type "rien"'
```



Antislash

Que faire si on a besoin des 2 types de guillemets ?

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Antislash

Que faire si on a besoin des 2 types de guillemets ?

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"
► Chaînes de
caractères

Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> ' "Aujourd'hui" le ciel est beau'
```

```
File "<stdin>", line 1
```

```
' "Aujourd'hui" le ciel est beau'
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> ' "Aujourd\'hui" le ciel est beau'
```

```
' "Aujourd\'hui" le ciel est beau'
```



Antislash

Que faire si on a besoin des 2 types de guillemets ?

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"
► Chaînes de
caractères

Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> ' "Aujourd'hui" le ciel est beau'
```

```
File "<stdin>", line 1
```

```
' "Aujourd'hui" le ciel est beau'
```

```
^
```

```
SyntaxError: invalid syntax
```

```
>>> ' "Aujourd\'hui" le ciel est beau'
```

```
' "Aujourd\'hui" le ciel est beau'
```

Autres usages

- `\n` pour les sauts de ligne
- `\t` pour les tabulations
- `\\` pour les antislashes



Raw string

L'écriture de certaines chaînes de caractères peut vite devenir très compliquée (chemins d'accès à des fichiers par exemple). Il y a donc une syntaxe spéciale en Python pour que les “\” soient interprétés comme tels.

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type “rien”

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Raw string

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

L'écriture de certaines chaînes de caractères peut vite devenir très compliquée (chemins d'accès à des fichiers par exemple). Il y a donc une syntaxe spéciale en Python pour que les “\” soient interprétés comme tels.

Il suffit d'ajouter un “r” devant la chaîne de caractères. Cela s'appelle une “raw string”.

```
>>> print("C:\Images\noe1")  
C:\Images  
oe1
```

```
>>> print(r"C:\Images\noe1")  
C:\Images\noe1
```



Mise en page

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python propose aussi une syntaxe afin de garder la mise en page d'une chaîne de caractères. Il faut que celle-ci commence et finisse par 3 guillemets.



Mise en page

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python propose aussi une syntaxe afin de garder la mise en page d'une chaîne de caractères. Il faut que celle-ci commence et finisse par 3 guillemets.

```
>>> print("""ceci est une ligne
          ceci est une autre ligne

          ceci est une tabulation""")

ceci est une ligne
ceci est une autre ligne

ceci est une tabulation
```



Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Voici la liste des opérations utilisables sur des chaînes de caractères :

Opérateur	Signification
+	Concaténation
*	Répétition
in	Inclusion d'une chaîne dans une autre
not in	Non-inclusion
[i]	i-ième caractère d'une chaîne
[i:j]	Caractères entre la i-ième et la j-ième positions



Fonctions courantes

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Fonction	Signification
chaine.count(s)	compte le nombre d'occurrences de "s"
chaine.find(s)	retourne la position du premier "s" de la chaîne, ou -1 s'il n'apparaît pas
chaine.isalpha()	retourne True si la chaîne est uniquement composée de lettres
chaine.isdigit()	retourne True si la chaîne est uniquement composée de chiffres
chaine.replace(old,new)	remplace toutes les occurrences de "old" par "new"
chaine.split(sep)	découpe la chaîne au niveau de "sep"
chaine.strip(s)	supprime les espaces au début et à la fin de la chaîne ainsi que les "s"
chaine.upper()	mets le texte en majuscule
chaine.startswith(s)	retourne True si la chaîne commence par "s"



Formatage de texte

Depuis la version 2.6, il existe 2 façons différentes de formater des chaînes de caractères :

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

► Chaînes de
caractères

Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Formatage de texte

Depuis la version 2.6, il existe 2 façons différentes de formater des chaînes de caractères :

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
► Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> "j'affiche la valeur %i" %(3)
"j'affiche la valeur 3"
```

```
>>> "j'affiche la valeur {}".format(3)
"j'affiche la valeur 3"
```



Formatage de texte

Depuis la version 2.6, il existe 2 façons différentes de formater des chaînes de caractères :

```
>>> "j'affiche la valeur %i" %(3)
"j'affiche la valeur 3"
```

```
>>> "j'affiche la valeur {}".format(3)
"j'affiche la valeur 3"
```

Remarque

Ces méthodes ne sont pas strictement identiques, mais dans le cadre de notre cours, on pourra considérer qu'elles le sont.

▶ [Verbose mode](#)

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
▶ Chaînes de
caractères
Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Séquences

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les séquences sont un ensemble de types de données, on y retrouve :



Séquences

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les séquences sont un ensemble de types de données, on y retrouve :

- list
- tuples
- range



Séquences

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les séquences sont un ensemble de types de données, on y retrouve :

- list
- tuples
- range

On peut également considérer les chaînes de caractères comme un type particulier de séquences.



Généralités

Les séquences sont :

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Généralités

Les séquences sont :

- itérables : elles peuvent retourner les éléments qui les composent les uns à la suite des autres

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Généralités

Les séquences sont :

- itérables : elles peuvent retourner les éléments qui les composent les uns à la suite des autres
- indexables : elles peuvent retourner un élément particulier sans parcourir l'intégralité de leurs éléments

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Généralités

Les séquences sont :

- itérables : elles peuvent retourner les éléments qui les composent les uns à la suite des autres
- indexables : elles peuvent retourner un élément particulier sans parcourir l'intégralité de leurs éléments

Indexation

L'indexation des séquences commence à 0.

- $s[0]$ \Rightarrow premier élément de la séquence
- $s[1]$ \Rightarrow second élément de la séquence

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Généralités

Les séquences sont :

- itérables : elles peuvent retourner les éléments qui les composent les uns à la suite des autres
- indexables : elles peuvent retourner un élément particulier sans parcourir l'intégralité de leurs éléments

Indexation

L'indexation des séquences commence à 0.

- `s[0]` \Rightarrow premier élément de la séquence
- `s[1]` \Rightarrow second élément de la séquence

```
>>> s = "12345"  
>>> s[0]  
'1'
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Opérations

Opérateur	Signification
<code>x in s</code>	<code>x</code> est dans <code>s</code>
<code>x not in s</code>	<code>x</code> n'est pas dans <code>s</code>
<code>s + t</code>	concaténation
<code>s * n</code>	<code>s</code> ajouté <code>n</code> fois à lui-même
<code>s[i]</code>	élément à la <code>i</code> -ième position
<code>s[i:j]</code>	éléments de la <code>i</code> -ième place à la <code>j</code> -ième place
<code>s[i:j:k]</code>	éléments de la <code>i</code> -ième place à la <code>j</code> -ième place avec un pas de <code>k</code>
<code>len(s)</code>	nombre d'éléments de <code>s</code>
<code>min(s)</code>	plus petit élément de <code>s</code> (si possible)
<code>sum(s)</code>	somme des éléments de <code>s</code> (si possible)
<code>s.index(x,i,j)</code>	index de la première occurrence de <code>x</code> entre les <code>i+1</code> et <code>j+1</code> -ièmes éléments

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Catégories de séquences

Il existe 2 catégories de séquences :

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Catégories de séquences

Il existe 2 catégories de séquences :

- séquences immuables : les éléments ne peuvent pas être modifiés

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Catégories de séquences

Il existe 2 catégories de séquences :

- séquences immuables : les éléments ne peuvent pas être modifiés
- séquences modifiables : les éléments peuvent être modifiés

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Catégories de séquences

Il existe 2 catégories de séquences :

- séquences immuables : les éléments ne peuvent pas être modifiés
- séquences modifiables : les éléments peuvent être modifiés

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> seq = "abc"
```

```
>>> id(seq)
```

```
139981844948616
```

```
>>> seq = seq+"de"
```

```
>>> id(seq)
```

```
139981767557728
```

```
>>> seq = []
```

```
>>> id(seq)
```

```
140407931682952
```

```
>>> seq.append(0)
```

```
>>> id(seq)
```

```
140107931682952
```



Tuples

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Un tuple est un tableau d'objets de n'importe quel type. Les tuples sont immuables.



Tuples

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Un tuple est un tableau d'objets de n'importe quel type. Les tuples sont immuables.

```
>>> t = ("abcde", 12345, Decimal(2.35557))
>>> t
('abcde', 12345, Decimal('
2.355570000000000016370904631912708282470703125'))
```

Les tuples sont toujours représentés avec des parenthèses.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Il est possible d'imbriquer des tuples.

```
>>> t = (123, "abcde", 18.1)
```

```
>>> t
```

```
(123, 'abcde', 18.1)
```

```
>>> u = (t, "xyz")
```

```
>>> u
```

```
((123, 'abcde', 18.1), 'xyz')
```



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"
Chaînes de
caractères

► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pour créer un tuple vide, on l'initialise juste avec des parenthèses ou avec $t = \text{tuple}()$.

```
>>> t = ()
```

```
>>> t
```

```
()
```



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pour créer un tuple vide, on l'initialise juste avec des parenthèses ou avec $t = \text{tuple}()$.
- Si un tuple ne contient qu'un élément il faut quand même mettre une virgule sinon Python convertit le tuple en type de son unique élément.

```
>>> t = () | >>> t = (1) | >>> t = (1,)
>>> t | >>> type(t) | >>> t
() | <class 'int'> | (1,)
```



Listes

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes sont similaires aux tuples, mais il est possible de modifier / ajouter / supprimer des éléments d'une liste sans créer un nouvel objet. Les listes sont modifiables.



Listes

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes sont similaires aux tuples, mais il est possible de modifier / ajouter / supprimer des éléments d'une liste sans créer un nouvel objet. Les listes sont modifiables.

```
>>> l = ["abcde", 12345, 3.14]
>>> l
['abcde', 12345, 3.14]
```



Listes

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes sont similaires aux tuples, mais il est possible de modifier / ajouter / supprimer des éléments d'une liste sans créer un nouvel objet. Les listes sont modifiables.

```
>>> l = ["abcde", 12345, 3.14]
>>> l
['abcde', 12345, 3.14]
```

Les listes sont représentées avec des crochets.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Similairement aux tuples :

- Les listes sont imbriquables

```
>>> l = ["abcde", 12345, 3.14]
>>> m = [1, 3.845]
>>> m
[ ['abcde', 12345, 3.14], 3.845]
```



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
▶ Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Similairement aux tuples :

- Les listes sont imbriquables
- Une liste vide s'initialise avec des crochets ou avec $l = list()$

```
>>> l = ["abcde", 12345, 3.14]
```

```
>>> m = [1, 3.845]
```

```
>>> m
```

```
[['abcde', 12345, 3.14], 3.845]
```

```
>>> l = []
```

```
>>> l
```

```
[]
```




Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Du fait de leurs caractères modifiables, les listes possèdent quelques opérateurs supplémentaires :

Opérateur	Signification
$l[i] = x$	i-ème élément remplacé par x
$l[i:j] = m$	les éléments entre les positions i et j sont remplacés par la liste m
$del(l[i:j])$	supprime les éléments entre les positions i et j
$l += m$	ajoute la liste m à l
$l *= n$	ajoute n-1 fois l à elle-même



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"
Chaînes de
caractères

► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les listes bénéficient aussi de fonctions supplémentaires :

Fonction	Signification
<code>l.append(x)</code>	ajoute <code>x</code> à la fin de <code>l</code>
<code>l.extend(m)</code>	ajoute la liste <code>m</code> à la fin de <code>l</code>
<code>l.insert(i,x)</code>	insert <code>x</code> à la position <code>i</code>
<code>l.remove(x)</code>	supprime tous les <code>x</code> dans <code>l</code>
<code>l.pop(i)</code>	retourne et supprime de la liste l'élément à la position <code>i</code>
<code>l.clear()</code>	vide la liste
<code>l.reverse()</code>	inverse l'ordre des éléments
<code>l.copy()</code>	copie la liste
<code>l.sort()</code>	trie les éléments de la liste par ordre croissant



Ranges

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

► Séquences

Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les ranges sont des séquences immuables de nombres entiers. Elles sont définies à partir de bornes initiale, finale et d'un pas.



Ranges

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les ranges sont des séquences immuables de nombres entiers. Elles sont définies à partir de bornes initiale, finale et d'un pas.

- La borne initiale par défaut est 0
- Le pas par défaut est 1
- Pour visualiser une range on la convertit en liste



Ranges

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
► Séquences
Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les ranges sont des séquences immuables de nombres entiers. Elles sont définies à partir de bornes initiale, finale et d'un pas.

- La borne initiale par défaut est 0
- Le pas par défaut est 1
- Pour visualiser une range on la convertit en liste

```
>>> r = list(range(5))
```

```
>>> r
```

```
[0, 1, 2, 3, 4]
```



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
► Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Dictionnaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les dictionnaires sont semblables aux listes, mais sont différents pour le référencement des valeurs. Au lieu d'associer un élément à sa position, un dictionnaire lui associe une clé.



Dictionnaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les dictionnaires sont semblables aux listes, mais sont différents pour le référencement des valeurs. Au lieu d'associer un élément à sa position, un dictionnaire lui associe une clé.

Une clé est un élément de type immuable (entier, réel, chaîne, tuple).



Dictionnaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les dictionnaires sont semblables aux listes, mais sont différents pour le référencement des valeurs. Au lieu d'associer un élément à sa position, un dictionnaire lui associe une clé.

Une clé est un élément de type immuable (entier, réel, chaîne, tuple).

```
>>> dico = {"bleu": "0000FF", "vert": "00FF00", "rouge":  
            "FF0000"}  
>>> dico["bleu"]  
'0000FF'
```



Remarques

- Un dictionnaire n'est pas triable (ses éléments ne sont pas ordonnés).

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Un dictionnaire n'est pas triable (ses éléments ne sont pas ordonnés).
- On accède à un élément d'un dictionnaire pas sa clé.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Un dictionnaire n'est pas triable (ses éléments ne sont pas ordonnés).
- On accède à un élément d'un dictionnaire pas sa clé.
- On peut ajouter / modifier / supprimer un élément du dictionnaire : les dictionnaires sont modifiables.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

► Dictionnaires

Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Un dictionnaire n'est pas triable (ses éléments ne sont pas ordonnés).
- On accède à un élément d'un dictionnaire pas sa clé.
- On peut ajouter / modifier / supprimer un élément du dictionnaire : les dictionnaires sont modifiables.
- On initialise un dictionnaire vide par des accolades ou avec $d = dict()$.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
► Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Un dictionnaire n'est pas triable (ses éléments ne sont pas ordonnés).
- On accède à un élément d'un dictionnaire pas sa clé.
- On peut ajouter / modifier / supprimer un élément du dictionnaire : les dictionnaires sont modifiables.
- On initialise un dictionnaire vide par des accolades ou avec $d = dict()$.

```
>>> dico["jaune"] = "00FFFF"
>>> dico
{'bleu': '0000FF', 'vert': '00FF00', 'rouge': 'FF0000', 'jaune': '00FFFF'}
```



Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
► Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les opérations suivantes s'appliquent aux dictionnaires :

Opérateur	Signification
<code>x in d</code>	x est dans une des clés de d
<code>x not in d</code>	x n'est pas dans une des clés de d
<code>d[c]</code>	retourne l'élément associé à la clé c
<code>len(d)</code>	nombre d'éléments de d
<code>del(d[c])</code>	supprime l'élément associé à la clé c



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
► Dictionnaires
Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les fonctions suivantes sont utilisables sur les dictionnaires :

Fonction	Signification
<code>d.items()</code>	retourne une liste des couples (clé,valeur) du dictionnaire
<code>list(d.keys())</code>	retourne une liste des clés du dictionnaire
<code>list(d.values())</code>	retourne une liste des valeurs du dictionnaire
<code>d.update(d2)</code>	ajoute à d2 les valeurs de d qui n'étaient pas déjà dans d2
<code>d.clear()</code>	vide le dictionnaire



Déroulement de la présentation

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations**
 - Types
 - Les types numériques
 - Booléens
 - Type "rien"
 - Chaînes de caractères
 - Séquences
 - Dictionnaires
 - Ensembles
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Ensembles

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les ensembles (set) sont des collections non ordonnées d'objets. Ils sont similaires aux listes ou aux dictionnaires et possèdent certaines propriétés des types numériques.



Ensembles

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les ensembles (set) sont des collections non ordonnées d'objets. Ils sont similaires aux listes ou aux dictionnaires et possèdent certaines propriétés des types numériques.

```
>>> a = set("ensg geomatique")
>>> a
{'q', 'g', 'e', 'a', 't', 'u', 'm', 'n', 'o', 'i', ' ', 's'}
```



Remarques

Les ensembles :

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).
- sont modifiables (ajout / modification / suppression d'éléments).



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).
- sont modifiables (ajout / modification / suppression d'éléments).
- peuvent contenir tout type d'objets.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).
- sont modifiables (ajout / modification / suppression d'éléments).
- peuvent contenir tout type d'objets.
- supportent les propriétés mathématiques des ensembles.



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).
- sont modifiables (ajout / modification / suppression d'éléments).
- peuvent contenir tout type d'objets.
- supportent les propriétés mathématiques des ensembles.
- ne sont pas indexables (impossible de récupérer un élément à partir de sa position).



Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles :

- sont itérables (il est possible de retourner les éléments qui els composent les uns à la suite des autres).
- sont modifiables (ajout / modification / suppression d'éléments).
- peuvent contenir tout type d'objets.
- supportent les propriétés mathématiques des ensembles.
- ne sont pas indexables (impossible de récupérer un élément à partir de sa position).
- **ne contiennent pas d'objets en double.**



Opérations

Les ensembles supportent les opérations suivantes :

Opérateur	Signification
$a \& b$	intersection
$a \mid b$	union
$a - b$	différence
$a \hat{=} b$	différence symétrique

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques

Booléens
Type "rien"

Chaînes de
caractères

Séquences
Dictionnaires

► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Opérations

Présentation
du module

Présentation
de Python

Types et
Opérations

Types

Les types
numériques

Booléens

Type "rien"

Chaînes de
caractères

Séquences

Dictionnaires

► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Les ensembles supportent les opérations suivantes :

Opérateur	Signification
$a \& b$	intersection
$a b$	union
$a - b$	différence
$a \wedge b$	différence symétrique

Différences

- $a - b$ retourne les éléments qui sont dans a et pas dans b .
- $a \wedge b$ retourne les éléments qui sont dans a ou dans b mais pas dans les deux.



Exemples

Présentation
du module

```
>>> a = set("ensg geomatique")
```

```
>>> a
```

```
{'q', 'g', 'e', 'a', 't', 'u', 'm', 'n', 'o', 'i', ' ', 's'}
```

Présentation
de Python

Types et
Opérations

```
>>> b = set("je suis un ensemble")
```

```
>>> b
```

```
{'l', 'e', 'm', 's', 'n', 'b', 'i', 'j', ' ', 'u'}
```

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

```
>>> a & b
```

```
{'e', 'u', 'm', 'n', 'i', ' ', 's'}
```

Syntaxe

```
>>> a - b
```

```
{'o', 'q', 't', 'a', 'g'}
```

Modules et
fonctions

```
>>> a ^ b
```

```
{'q', 'l', 'g', 'a', 't', 'b', 'o', 'j'}
```

Conclusion

References



Fonctions

Les ensembles supportent les fonctions suivantes :

Fonction	Signification
<code>s.add(x)</code>	ajoute <code>x</code> s'il n'est pas déjà présent
<code>s.remove(x)</code>	enlève <code>x</code> s'il est présent dans l'ensemble
<code>s.copy()</code>	copie <code>s</code>
<code>s.clear()</code>	vide <code>s</code>
<code>s.issubset(t)</code>	retourne <code>True</code> si <code>s</code> est inclus dans <code>t</code>
<code>s.isdisjoint(t)</code>	retourne <code>True</code> si l'intersection de <code>s</code> et <code>t</code> est vide
<code>s.pop()</code>	retire un élément aléatoirement
<code>s.difference(t)</code>	renvoie la différence entre <code>s</code> et <code>t</code>
<code>s.difference_update(t)</code>	modifie <code>s</code> pour qu'il devienne la différence entre <code>s</code> et <code>t</code>

Présentation
du module

Présentation
de Python

Types et
Opérations

Types
Les types
numériques
Booléens
Type "rien"
Chaînes de
caractères
Séquences
Dictionnaires
► Ensembles

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

► Syntaxe

Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations
- 4 Syntaxe**
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Syntaxe

Maintenant que nous maîtrisons les différents types de données, nous allons nous intéresser à la rédaction de programmes en Python.

Présentation
du module

Présentation
de Python

Types et
Opérations

► Syntaxe

Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Syntaxe

Présentation
du module

Présentation
de Python

Types et
Opérations

► Syntaxe

Structure du
code
Affectation
Tests
Boucles

Modules et
fonctions

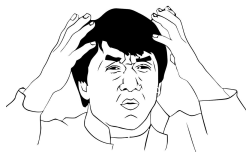
Fichiers et
itérateurs

Conclusion

References

Maintenant que nous maîtrisons les différents types de données, nous allons nous intéresser à la rédaction de programmes en Python.

Les blocs sont définis par un ensemble d'instructions consécutives situés à un même niveau d'indentation. Le retour à un niveau d'indentation inférieur marque la fin d'un bloc.





Syntaxe

Présentation
du module

Présentation
de Python

Types et
Opérations

► Syntaxe

Structure du
code
Affectation
Tests
Boucles

Modules et
fonctions

Fichiers et
itérateurs

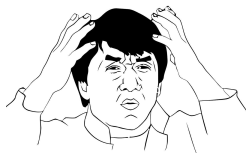
Conclusion

References

Maintenant que nous maîtrisons les différents types de données, nous allons nous intéresser à la rédaction de programmes en Python.

Les blocs sont définis par un ensemble d'instructions consécutives situés à un même niveau d'indentation. Le retour à un niveau d'indentation inférieur marque la fin d'un bloc.

L'instruction qui débute le bloc se termine par ":".





Déroulement de la présentation

Présentation
du module

1 Présentation du module

Présentation
de Python

2 Présentation de Python

Types et
Opérations

3 Types et Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

4 **Syntaxe**

- Structure du code
- Affectation
- Tests
- Boucles

5 Modules et fonctions

6 Fichiers et itérateurs

7 Conclusion



Structure du code

Contrairement à d'autres langages (C++, Java, ...), il n'y a pas de caractère de fin d'instruction en Python. Chaque instruction est donc écrite sur une ligne différente. Le recul d'une indentation marque la fin d'un bloc.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Structure du code

Contrairement à d'autres langages (C++, Java, ...), il n'y a pas de caractère de fin d'instruction en Python. Chaque instruction est donc écrite sur une ligne différente. Le recul d'une indentation marque la fin d'un bloc.

```
if condition:
    instruction1
    instruction2
instruction3
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Structure du code

Contrairement à d'autres langages (C++, Java, ...), il n'y a pas de caractère de fin d'instruction en Python. Chaque instruction est donc écrite sur une ligne différente. Le recul d'une indentation marque la fin d'un bloc.

```
if condition:
    instruction1
    instruction2
instruction3
```

Remarque

Les espaces et les sauts de ligne ne sont pas pris en compte par l'interpréteur Python.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

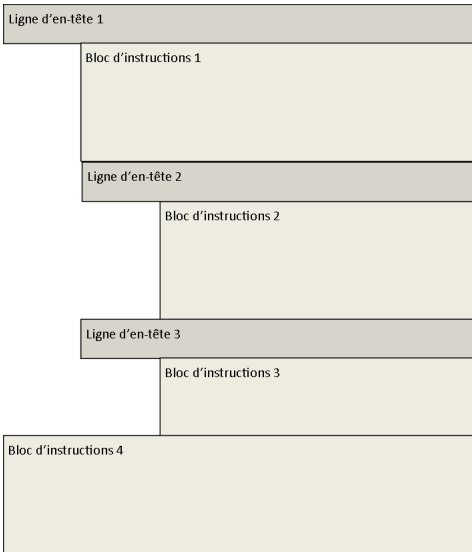
Fichiers et
itérateurs

Conclusion

References



Structure du code



Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Commentaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python permet d'ajouter des commentaires dans le code, qui sont ignorés par l'interpréteur. Il en existe 2 types principaux :



Commentaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python permet d'ajouter des commentaires dans le code, qui sont ignorés par l'interpréteur. Il en existe 2 types principaux :

- commentaire simple : commence par “#” et termine à la fin de la ligne



Commentaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Python permet d'ajouter des commentaires dans le code, qui sont ignorés par l'interpréteur. Il en existe 2 types principaux :

- commentaire simple : commence par “#” et termine à la fin de la ligne
- les docstrings : chaîne de caractères commençant et finissant par 3 guillemets



Commentaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Structure du
code

Affectation
Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
def nb_points_sub_domain(points):  
    """  
    Fonction qui calcule le nombre de points appartenant  
    à un sous-domaine précis, pour une liste de points donnée  
  
    :param points: liste de points  
    :return: nombre de points appartenant au sous-domaine  
    """  
    nb_points = 0  
    # On parcourt la liste de points et on cherche  
    # leur appartenance au cercle trigonométrique  
    for point in points:  
        if is_inside_circle(point[0], point[1]):  
            nb_points += 1  
    print (nb_points)  
    return nb_points
```



Déroulement de la présentation

Présentation
du module

1 Présentation du module

Présentation
de Python

5 Modules et fonctions

Types et
Opérations

2 Présentation de Python

6 Fichiers et itérateurs

Syntaxe

7 Conclusion

Structure du
code

3 Types et Opérations

► Affectation

Tests

Boucles

4 **Syntaxe**

● Structure du code

Fichiers et
itérateurs

● Affectation

Conclusion

● Tests

References

● Boucles



Affectation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

► Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pour affecter une valeur à une variable, on utilise le signe “=” .

```
>>> a = 1
>>> a
1
```



Affectation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

► Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

- Pour affecter une valeur à une variable, on utilise le signe “=”.
- Il est également possible de réaliser plusieurs affectations en une seule instruction.

```
>>> a = 1
```

```
>>> a
```

```
1
```

```
>>> a,b = 1,2
```

```
>>> a
```

```
1
```

```
>>> b
```

```
2
```

```
>>> a = b = 1
```

```
>>> a
```

```
1
```

```
>>> b
```

```
1
```



Affectations augmentées

Dans une seule instruction, Python peut réaliser une affectation et modifier la valeur de la variable cible ensuite. Cela s'appelle une affectation augmentée.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

► Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Affectations augmentées

Dans une seule instruction, Python peut réaliser une affectation et modifier la valeur de la variable cible ensuite. Cela s'appelle une affectation augmentée.

Ainsi, les instructions suivantes sont équivalentes :

```
>>> a = 3
```

```
>>> a = a + 3
```

```
>>> a
```

```
6
```

```
>>> a += 3
```

```
>>> a
```

```
>>> a
```

```
6
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

► Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Affectations augmentées

Dans une seule instruction, Python peut réaliser une affectation et modifier la valeur de la variable cible ensuite. Cela s'appelle une affectation augmentée.

Ainsi, les instructions suivantes sont équivalentes :

```
>>> a = 3
```

```
>>> a = a + 3
```

```
>>> a
```

```
6
```

```
>>> a = 3
```

```
>>> a += 3
```

```
>>> a
```

```
6
```

Remarque

Les affectations augmentées fonctionnent avec tous les types de données et tous les opérateurs, il suffit de rajouter un “=” derrière l’opérateur : +=, *=, %=, ...

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

► Affectation

Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Déroulement de la présentation

Présentation
du module

1 Présentation du module

Présentation
de Python

5 Modules et fonctions

Types et
Opérations

2 Présentation de Python

6 Fichiers et itérateurs

Syntaxe

3 Types et Opérations

7 Conclusion

Structure du
code
Affectation
► Tests
Boucles

4 **Syntaxe**

● Structure du code

● Affectation

● **Tests**

● Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Tests

“if”

Les tests permettent de réaliser différentes instructions selon une condition. On teste cette condition avec le mot clé “if”.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> x = 1
>>> if (x > 0):
    print(x)
```

1



Tests

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

“if”

Les tests permettent de réaliser différentes instructions selon une condition. On teste cette condition avec le mot clé “if”.

“else”

S'il faut réaliser d'autres instructions dans le cas où la condition n'est pas vérifiée, on utilise le mot clé “else”.

```
>>> x = 1
>>> if (x > 0):
    print(x)
```

1

```
>>> if (x < 0):
    print("x négatif")
else:
    print("x positif")
```

x positif



Tests

“elif”

S'il faut réaliser plusieurs tests, ou si on veut différencier plus que 2 possibilités pour la condition, on peut utiliser le mot clé “elif”.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Tests

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

“elif”

S'il faut réaliser plusieurs tests, ou si on veut différencier plus que 2 possibilités pour la condition, on peut utiliser le mot clé “elif”.

```
>>> if (x > 5):  
    print ("x > 5")  
elif (x > 0):  
    print ("0 < x < 5")  
else:  
    print ("x < 0")
```

```
0 < x < 5
```

Indentation

En Python, la structure étant caractérisée par l'indentation, il faut penser à indenter les instructions qui suivent chaque “if”, “elif” ou “else”.



Simplification des conditions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible de simplifier l'écriture de certaines conditions lorsque la valeur est comparée à None, un booléen ou 0.



Simplification des conditions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible de simplifier l'écriture de certaines conditions lorsque la valeur est comparée à None, un booléen ou 0.

Valeur	Test	Test simplifié



Simplification des conditions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible de simplifier l'écriture de certaines conditions lorsque la valeur est comparée à None, un booléen ou 0.

Valeur	Test	Test simplifié
None	if x == None	if not x
None	if x != None	if x



Simplification des conditions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible de simplifier l'écriture de certaines conditions lorsque la valeur est comparée à None, un booléen ou 0.

Valeur	Test	Test simplifié
None	if x == None	if not x
None	if x != None	if x
Booléen	if x == True	if x
Booléen	if x == False	if not x



Simplification des conditions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible de simplifier l'écriture de certaines conditions lorsque la valeur est comparée à None, un booléen ou 0.

Valeur	Test	Test simplifié
None	if x == None	if not x
None	if x != None	if x
Booléen	if x == True	if x
Booléen	if x == False	if not x
Zéro	if x == 0	if not x
Zéro	if x != 0	if x



Tests

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

“pass”

Lors qu'aucune instruction ne doit être réalisée, selon une certaine condition, on utilise le mot clé “pass”.



Tests

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

► Tests

Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

“pass”

Lors qu'aucune instruction ne doit être réalisée, selon une certaine condition, on utilise le mot clé “pass”.

```
>>> if (not x):  
    pass  
    elif (x > 0):  
        print ("x positif")  
    else:  
        print ("x négatif")
```

```
x positif
```



Déroulement de la présentation

Présentation
du module

1 Présentation du module

Présentation
de Python

5 Modules et fonctions

Types et
Opérations

2 Présentation de Python

6 Fichiers et itérateurs

Syntaxe

3 Types et Opérations

7 Conclusion

Structure du
code
Affectation
Tests
► Boucles

4 **Syntaxe**

- Structure du code
- Affectation
- Tests
- **Boucles**

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucles

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les boucles permettent d'exécuter plusieurs fois un bloc d'instructions. En Python les mots clés "while" et "for" sont utilisés pour faire des boucles.



Boucles

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les boucles permettent d'exécuter plusieurs fois un bloc d'instructions. En Python les mots clés "while" et "for" sont utilisés pour faire des boucles.

Les instructions à l'intérieur d'une boucle sont indentées. C'est le retour à l'indentation initiale qui marque la fin de la boucle.



Boucle while

Définition

Le boucle while permet de réaliser un bloc d'instructions tant que la condition est réalisée. Il est possible de rajouter un "else" qui contiendra des instructions réalisées une fois que les itérations de la boucle sont terminées.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle while

Définition

Le boucle while permet de réaliser un bloc d'instructions tant que la condition est réalisée. Il est possible de rajouter un "else" qui contiendra des instructions réalisées une fois que les itérations de la boucle sont terminées.

```
>>> x = 0
>>> while (x < 3):
    print(x)
    x += 1
else:
    print("boucle terminée")

0
1
2
boucle terminée
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle while

Définition

Le boucle while permet de réaliser un bloc d'instructions tant que la condition est réalisée. Il est possible de rajouter un "else" qui contiendra des instructions réalisées une fois que les itérations de la boucle sont terminées.

```
>>> x = 0
>>> while (x < 3):
    print(x)
    x += 1
else:
    print("boucle terminée")

0
1
2
boucle terminée
```

Attention

Attention aux boucles infinies (while True) !

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle for

Définition

Le boucle for permet d'itérer une séquence (liste, tuple, range, dictionnaire, ...) et d'exécuter un bloc d'instructions pour tous les éléments de cette séquence. Comme pour la boucle for, il est possible d'ajouter un "else" à la fin.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle for

Définition

Le boucle for permet d'itérer une séquence (liste, tuple, range, dictionnaire, ...) et d'exécuter un bloc d'instructions pour tous les éléments de cette séquence. Comme pour la boucle for, il est possible d'ajouter un "else" à la fin.

```
>>> x = 0
>>> for i in range(3):
    print (x)
    x += 1
else:
    print ("fin de l'exécution")

0
1
2
fin de l'exécution
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle for

Il est possible d'itérer sur les éléments d'une chaîne de caractères, des tuples, les clés d'un dictionnaire, ...

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Boucle for

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible d'itérer sur les éléments d'une chaîne de caractères, des tuples, les clés d'un dictionnaire, ...

```
>>> s = "ensg"  
>>> for lettre in s:  
    print(lettre)
```

e

n

s

g



Boucle for

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible d'itérer sur les éléments d'une chaîne de caractères, des tuples, les clés d'un dictionnaire, ...

```
>>> s = "ensg"  
>>> for lettre in s:  
        print(lettre)
```

e
n
s
g

```
>>> l = [(1,2), (3,4), (5,6)]  
>>> for (i,j) in l:  
        print(i,j)
```

1 2
3 4
5 6



Boucle for

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Il est possible d'itérer sur les éléments d'une chaîne de caractères, des tuples, les clés d'un dictionnaire, ...

```
>>> dico
{'bleu': '0000FF', 'vert': '00FF00', 'rouge': 'FF0000', '
    jaune': '00FFFF'}
>>> for clef in dico:
    print(clef, dico[clef])
```

```
bleu 0000FF
vert 00FF00
rouge FF0000
jaune 00FFFF
```



Les sorties de boucle

En Python, il existe 2 syntaxes différentes permettant de sortir d'une boucle avant la fin de son exécution normale. On parle de débranchements.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Les sorties de boucle

En Python, il existe 2 syntaxes différentes permettant de sortir d'une boucle avant la fin de son exécution normale. On parle de débranchements.

- “break” permet de sortir définitivement d'une boucle

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Les sorties de boucle

En Python, il existe 2 syntaxes différentes permettant de sortir d'une boucle avant la fin de son exécution normale. On parle de débranchements.

- “break” permet de sortir définitivement d'une boucle
- “continue” permet de retourner au début des instructions de la boucle

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Les sorties de boucle

En Python, il existe 2 syntaxes différentes permettant de sortir d'une boucle avant la fin de son exécution normale. On parle de débranchements.

- “break” permet de sortir définitivement d'une boucle
- “continue” permet de retourner au début des instructions de la boucle

Clause “else”

Si l'on veut exécuter un groupe d'instructions après qu'une boucle se soit terminée correctement, on peut utiliser un “else”.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



Les sorties de boucle

En Python, il existe 2 syntaxes différentes permettant de sortir d'une boucle avant la fin de son exécution normale. On parle de débranchements.

- “break” permet de sortir définitivement d'une boucle
- “continue” permet de retourner au début des instructions de la boucle

Clause “else”

Si l'on veut exécuter un groupe d'instructions après qu'une boucle se soit terminée correctement, on peut utiliser un “else”.

“pass”

Il est aussi possible d'utiliser le mot cle “pass” dans les boucles.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References



“continue”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> for i in range(3):  
    print("toto")  
    continue  
    print("titi")  
else:  
    print("tata")
```

```
toto  
toto  
toto  
tata
```



“continue”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Placement de “continue”

Attention au placement de “continue”, cela peut entraîner des boucles infinies !

```
>>> x = 0
>>> while (x < 5):
        continue
        x += 1
```




“break”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> for x in range(5):  
    print(x)  
    if (x == 2):  
        break  
else:  
    print("boucle terminée")
```

0
1
2



“break”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation

Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> for x in range(5):  
    print(x)  
    if (x == 2):  
        break  
  
else:  
    print("boucle terminée")
```

0
1
2

Clause “else”

Dans le cas de l'utilisation d'un “break”, les instructions de la clause “else” ne sont pas lues.



Passer une liste au carré

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> liste = [1,2,3,4]
```

Comment passer tous les éléments de la liste au carré ?



Passer une liste au carré

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

```
>>> liste = [1,2,3,4]
```

Comment passer tous les éléments de la liste au carré ?
Solution évidente :

```
>>> liste2 = list()
>>> for element in liste:
    liste2.append(element ** 2)
```

```
>>> liste2
[1, 4, 9, 16]
```



Listes de compréhension

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes de compréhension sont une spécificité du Python. C'est la manière la plus rapide de créer une liste. Pour cela, il existe une syntaxe spécifique en Python, à partir d'une boucle for.



Listes de compréhension

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes de compréhension sont une spécificité du Python. C'est la manière la plus rapide de créer une liste. Pour cela, il existe une syntaxe spécifique en Python, à partir d'une boucle for.

```
>>> liste = [1,2,3,4]
>>> liste2 = [x ** 2 for x in liste]
>>> liste2
[1,4,9,16]
```



Listes de compréhension

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Structure du
code

Affectation
Tests

► Boucles

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

References

Définition

Les listes de compréhension sont une spécificité du Python. C'est la manière la plus rapide de créer une liste. Pour cela, il existe une syntaxe spécifique en Python, à partir d'une boucle for.

```
>>> liste = [1,2,3,4]
>>> liste2 = [x ** 2 for x in liste]
>>> liste2
[1,4,9,16]
```

Remarque

Cette syntaxe s'applique à tous les objets itérables.



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

► Modules et
fonctions

Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

6 Fichiers et itérateurs

7 Conclusion



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

- Fonctions
- Imports et modules
- Installer des packages

6 Fichiers et itérateurs

7 Conclusion



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Les fonctions sont des ensembles d'instructions appelables plusieurs fois dans un programme. Elles peuvent prendre certaines valeurs en entrée et retourner d'autres valeurs.



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Les fonctions sont des ensembles d'instructions appelables plusieurs fois dans un programme. Elles peuvent prendre certaines valeurs en entrée et retourner d'autres valeurs.

Remarques



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Les fonctions sont des ensembles d'instructions appelables plusieurs fois dans un programme. Elles peuvent prendre certaines valeurs en entrée et retourner d'autres valeurs.

Remarques

- Une fonction n'a pas forcément de paramètres ni de retours.



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Les fonctions sont des ensembles d'instructions appelables plusieurs fois dans un programme. Elles peuvent prendre certaines valeurs en entrée et retourner d'autres valeurs.

Remarques

- Une fonction n'a pas forcément de paramètres ni de retours.
- Les paramètres en entrée peuvent varier d'une exécution à une autre.



Fonctions

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Les fonctions sont des ensembles d'instructions appelables plusieurs fois dans un programme. Elles peuvent prendre certaines valeurs en entrée et retourner d'autres valeurs.

Remarques

- Une fonction n'a pas forcément de paramètres ni de retours.
- Les paramètres en entrée peuvent varier d'une exécution à une autre.
- L'ensemble des instructions à l'intérieur de la fonction s'appelle le corps.



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

- Une fonction commence par le mot clé “def”.



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

- Une fonction commence par le mot clé “def” .
- Le nom de la fonction est suivi de parenthèses.



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

- Une fonction commence par le mot clé “def” .
- Le nom de la fonction est suivi de parenthèses.
- Le corps de la fonction est indenté.



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

- Une fonction commence par le mot clé “def” .
- Le nom de la fonction est suivi de parenthèses.
- Le corps de la fonction est indenté.
- La fonction se termine lorsque l'indentation retourne à son niveau initial.



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

- Une fonction commence par le mot clé “def” .
- Le nom de la fonction est suivi de parenthèses.
- Le corps de la fonction est indenté.
- La fonction se termine lorsque l'indentation retourne à son niveau initial.
- Si la fonction possède un résultat, il est spécifié grâce au mot clé “return” .



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

```
def Nom_Fonction (paramètre_1, paramètre_2) :  
    instruction_1  
    instruction_2  
    instruction_3  
    return valeur_retournée
```



Syntaxe d'une fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

```
def Nom_Fonction (paramètre_1, paramètre_2) :  
    instruction_1  
    instruction_2  
    instruction_3  
    return valeur_retournée
```

```
def fibonacci(seuil):  
    """fib"""  
    n = [1,1]  
    while (len(n) < seuil):  
        n.append(n[len(n)-1] + n[len(n)-2])  
    return n
```



Appel de la fonction

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

On appelle une fonction en écrivant son nom suivi des parenthèses dans le programme.

```
>>> def my_function():  
        print("hello world")
```

```
>>> my_function()  
hello world
```



Valeurs retournées

Si la fonction retourne une valeur, on peut la récupérer de la façon suivante :

```
>>> def fonction():  
        return 0
```

```
>>> x = fonction()
```

```
>>> x  
0
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Valeurs retournées

Si la fonction retourne une valeur, on peut la récupérer de la façon suivante :

```
>>> def fonction():  
        return 0
```

```
>>> x = fonction()  
>>> x  
0
```

Lorsque la fonction retourne plusieurs valeurs, celles-ci sont écrites dans un tuple :

```
>>> def fonction():  
        return 0, "abcde", 3.1415
```

```
>>> t = fonction()  
>>> t  
(0, 'abcde', 3.1415)
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Valeurs retournées

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

On peut aussi récupérer les valeurs retournées de la façon suivante :

```
>>> def fonction():  
        return 0, "abcde", 3.1415
```

```
>>> r1,r2,r3 = fonction()  
>>> r1  
0  
>>> r2  
'abcde'  
>>> r3  
3.1415
```



Paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Les paramètres en entrée de la fonction s'ajoutent entre les parenthèses. Lors de l'appel de la fonction on précise les valeurs de ces paramètres.



Paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Les paramètres en entrée de la fonction s'ajoutent entre les parenthèses. Lors de l'appel de la fonction on précise les valeurs de ces paramètres.

```
>>> def plus(x,y):  
        p = x+y  
        return p  
  
>>> p = plus(1,2)  
>>> p  
3
```



Portée des variables

Définition

La portée d'une variable est la portion du programme à l'intérieur de laquelle elle référence une adresse mémoire (on dit qu'elle "existe").

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Portée des variables

Définition

La portée d'une variable est la portion du programme à l'intérieur de laquelle elle référence une adresse mémoire (on dit qu'elle "existe").

Remarques

- Les variables définies à l'intérieur d'une fonction n'existent qu'à l'intérieur de cette fonction.
- Les variables définies à l'extérieur d'une fonction ne sont pas modifiables par celle-ci.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Portée des variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

La portée d'une variable est la portion du programme à l'intérieur de laquelle elle référence une adresse mémoire (on dit qu'elle "existe").

Remarques

- Les variables définies à l'intérieur d'une fonction n'existent qu'à l'intérieur de cette fonction.
- Les variables définies à l'extérieur d'une fonction ne sont pas modifiables par celle-ci.

Pratique déconseillée

Les variables définies à l'extérieur d'une fonction sont utilisables par celle-ci si elles ont été déclarées avant.



Portée des variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Variable locale

Une variable est dite locale lorsque sa portée se limite à la fonction dans laquelle elle est définie.



Portée des variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Variable locale

Une variable est dite locale lorsque sa portée se limite à la fonction dans laquelle elle est définie.

Variable globale

Une variable est dite globale lorsqu'elle est visible dans tout le programme.



Portée des variables

```
>>> x = 1
>>> def test(y):
    z = x+y
    return z
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

```
>>> z = test(2)
>>> z
3
```

```
>>> y
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'y' is not defined
```

```
>>> def test2():
    x = 2
```

```
>>> test2()
>>> x
1
```



Mot clé “global”

Si l'on veut pouvoir modifier une variable globale depuis le corps d'une fonction, on peut utiliser le mot clé “global”.

```
>>> x = 1
>>> def test3():
    global x
    x = 2
```

```
>>> test3()
>>> x
2
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Mot clé “global”

Si l'on veut pouvoir modifier une variable globale depuis le corps d'une fonction, on peut utiliser le mot clé “global”.

```
>>> x = 1
>>> def test3():
    global x
    x = 2
```

```
>>> test3()
>>> x
2
```

Attention

On essaiera de s'affranchir de l'utilisation de ce mot clé, car cela revient à faire passer un paramètre caché à la fonction.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Fonctions incluses

En Python, les fonctions suivant les mêmes règles d'indentation que les boucles, il est possible de déclarer une fonction dans une autre fonction. La portée de la fonction fille est donc limitée à celle de sa fonction mère (la fonction fille n'existe qu'à l'intérieur de la fonction mère).

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Fonctions incluses

En Python, les fonctions suivant les mêmes règles d'indentation que les boucles, il est possible de déclarer une fonction dans une autre fonction. La portée de la fonction fille est donc limitée à celle de sa fonction mère (la fonction fille n'existe qu'à l'intérieur de la fonction mère).

```
>>> def fonction_mere():
    x=1
    def fonction_fille():
        print(x+1)
    fonction_fille()

>>> fonction_mere()
2

>>> fonction_fille()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'fonction_fille' is not defined
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :

❶ si la variable a été déclarée localement,



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :

- 1 si la variable a été déclarée localement,
- 2 si elle est déclarée dans le contexte d'inclusion (i.e. fonction englobante),



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :

- ❶ si la variable a été déclarée localement,
- ❷ si elle est déclarée dans le contexte d'inclusion (i.e. fonction englobante),
- ❸ s'il s'agit d'une variable globale,



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :

- 1 si la variable a été déclarée localement,
- 2 si elle est déclarée dans le contexte d'inclusion (i.e. fonction englobante),
- 3 s'il s'agit d'une variable globale,
- 4 si elle fait partie des objets intégrés au langage.



Règle LEGB

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

Lorsqu'une variable est appelée, Python regarde dans l'ordre :

- ❶ si la variable a été déclarée localement,
- ❷ si elle est déclarée dans le contexte d'inclusion (i.e. fonction englobante),
- ❸ s'il s'agit d'une variable globale,
- ❹ si elle fait partie des objets intégrés au langage.

On parle de règle LEGB pour *Local Enclosing Global Built-in*.



Paramètres par défaut

Lorsqu'une fonction est appelée de nombreuses fois avec les mêmes paramètres, il peut être intéressant de fixer des valeurs par défaut aux paramètres de la fonction afin de ne pas avoir à les re-spécifier à chaque appel.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Paramètres par défaut

Lorsqu'une fonction est appelée de nombreuses fois avec les mêmes paramètres, il peut être intéressant de fixer des valeurs par défaut aux paramètres de la fonction afin de ne pas avoir à les re-spécifier à chaque appel.

```
>>> def puissance (nombre, facteur=2):  
        x = nombre ** facteur  
        return x
```

```
>>> puissance(3)  
9  
>>> puissance(3,3)  
27
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Paramètres par défaut

Lorsqu'une fonction est appelée de nombreuses fois avec les mêmes paramètres, il peut être intéressant de fixer des valeurs par défaut aux paramètres de la fonction afin de ne pas avoir à les re-spécifier à chaque appel.

```
>>> def puissance (nombre, facteur=2):  
        x = nombre ** facteur  
        return x
```

```
>>> puissance(3)  
9  
>>> puissance(3,3)  
27
```

Position des paramètres

Attention à la position des paramètres, les paramètres avec des valeurs par défaut doivent toujours être placés après les autres paramètres.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Fonctions et types modifiables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

```
>>> def init_liste(n, liste):  
    for i in range(n):  
        liste[i] = 0  
    n += 1
```

```
>>> n = 2  
>>> liste = [1,2,3,4,5]  
>>> init_liste(n,liste)  
>>> n  
2  
>>> liste  
[0,0,3,4,5]
```



Fonctions et types modifiables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

```
>>> def init_liste(n, liste):  
    for i in range(n):  
        liste[i] = 0  
    n += 1
```

```
>>> n = 2  
>>> liste = [1,2,3,4,5]  
>>> init_liste(n,liste)  
>>> n  
2  
>>> liste  
[0,0,3,4,5]
```

Dans cette fonction, le paramètre `n` n'a pas été modifié. Ceci est dû au fait que `n` est d'un type immuable. On dit qu'il a été **passé en valeur**.



Fonctions et types modifiables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et modules
- Installer des packages

Fichiers et
itérateurs

Conclusion

References

```
>>> def init_liste(n, liste):  
    for i in range(n):  
        liste[i] = 0  
    n += 1
```

```
>>> n = 2  
>>> liste = [1,2,3,4,5]  
>>> init_liste(n,liste)  
>>> n  
2  
>>> liste  
[0,0,3,4,5]
```

Dans cette fonction, le paramètre `n` n'a pas été modifié. Ceci est dû au fait que `n` est d'un type immuable. On dit qu'il a été **passé en valeur**.

En revanche, la liste a été modifiée. ceci est dû au fait qu'elle est d'un type modifiable. On parle de **passage en paramètre**.



Passage des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Rappel

- Passer un paramètre à une fonction revient à affecter une valeur à une variable locale



Passage des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Rappel

- Passer un paramètre à une fonction revient à affecter une valeur à une variable locale
- L'attribution des noms des paramètres n'a pas de conséquence en dehors de la fonction



Passage des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Rappel

- Passer un paramètre à une fonction revient à affecter une valeur à une variable locale
- L'attribution des noms des paramètres n'a pas de conséquence en dehors de la fonction
- Modifier les valeurs d'un type modifiable dans une fonction a un impact sur l'objet en dehors de la fonction.



Passage des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions

Imports et
modules

Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Rappel

- Passer un paramètre à une fonction revient à affecter une valeur à une variable locale
- L'attribution des noms des paramètres n'a pas de conséquence en dehors de la fonction
- Modifier les valeurs d'un type modifiable dans une fonction a un impact sur l'objet en dehors de la fonction.
- L'ordre des paramètres a une importance.



Arguments positionnels variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Python autorise le fait de passer un nombre indéterminé de paramètres à une fonction. Pour cela on utilise la syntaxe “*n”. Les arguments sont passés à la fonction sous forme de tuple.



Arguments positionnels variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Python autorise le fait de passer un nombre indéterminé de paramètres à une fonction. Pour cela on utilise la syntaxe “*n”. Les arguments sont passés à la fonction sous forme de tuple.

```
>>> def pos_var_arg(*n):  
    print(n)  
    print(type(n))  
  
>>> pos_var_arg("abcde",123,[1,4,3.5])  
( 'abcde', 123, [1, 4, 3.5])  
<class 'tuple'>
```



Arguments nommés variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Il est possible de créer des fonctions acceptant un nombre indéterminé d'arguments nommés. La syntaxe est: `**options`. Les arguments sont passés à la fonction sous forme de dictionnaire.



Arguments nommés variables

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions

Imports et
modules

Installer des
packages

Fichiers et
itérateurs

Conclusion

References

```
def connect (** options):
```

```
    ''' connexion '''
```

```
    conn_params = {
```

```
        'host': options.get ( 'host' , '127.0.0.1') ,
```

```
        'port': options.get ( 'port' , 5432) ,
```

```
        'user': options.get ( 'user' , ' ' ) ,
```

```
        'pwd': options.get ( 'pwd' , ' ' ) ,
```

```
    }
```

```
    return conn_params
```

```
>>> connect()
```

```
{'pwd': ' ', 'host': '127.0.0.1', 'user': ' ', 'port': 5432}
```

```
>>> connect( host = '127.0.0.8' )
```

```
{'pwd': ' ', 'host': '127.0.0.8', 'user': ' ', 'port': 5432}
```

```
>>> connect( user = 'postgres', pwd = 'postgres' )
```

```
{'pwd': 'postgres', 'host': '127.0.0.1', 'user': 'postgres', 'port': 5432}
```



Ordre des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Si l'on souhaite combiner plusieurs types de paramètres, il faut respecter l'ordre suivant :



Ordre des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Si l'on souhaite combiner plusieurs types de paramètres, il faut respecter l'ordre suivant :

- 1 paramètres positionnels obligatoires



Ordre des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Si l'on souhaite combiner plusieurs types de paramètres, il faut respecter l'ordre suivant :

- ① paramètres positionnels obligatoires
- ② paramètres facultatifs avec valeurs par défaut



Ordre des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Si l'on souhaite combiner plusieurs types de paramètres, il faut respecter l'ordre suivant :

- ① paramètres positionnels obligatoires
- ② paramètres facultatifs avec valeurs par défaut
- ③ paramètres positionnels variables



Ordre des paramètres

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Si l'on souhaite combiner plusieurs types de paramètres, il faut respecter l'ordre suivant :

- ① paramètres positionnels obligatoires
- ② paramètres facultatifs avec valeurs par défaut
- ③ paramètres positionnels variables
- ④ paramètres nommés variables



Fonctions récursives

Définition

Une fonction récursive est une fonction qui s'appelle elle-même. L'exemple le plus classique est la fonction factorielle.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Fonctions récursives

Définition

Une fonction récursive est une fonction qui s'appelle elle-même. L'exemple le plus classique est la fonction factorielle.

```
def factorielle(n):  
    if (n == 0):  
        return 1  
    else:  
        return n*factorielle(n-1)  
  
>>> factorielle(5)  
120
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Fonctions récursives

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fonctions
Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Définition

Une fonction récursive est une fonction qui s'appelle elle-même. L'exemple le plus classique est la fonction factorielle.

```
def factorielle(n):  
    if (n == 0):  
        return 1  
    else:  
        return n*factorielle(n-1)  
  
>>> factorielle(5)  
120
```

Attention

Python n'autorise pas plus de 1000 appels récursifs
⇒ pour calculer `factorielle(1000)` il faut écrire une fonction non récursive.



Notions supplémentaires

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

- ▶ Fonctions
- Imports et
modules
- Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Fonctions anonymes: ▶ lambda

Générateurs: ▶ generateurs



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

- Fonctions
- Imports et modules
- Installer des packages

6 Fichiers et itérateurs

7 Conclusion



Modules

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Maintenant que nous savons écrire et appeler des fonctions au sein d'un même fichier .py, nous allons voir comment organiser notre programme en plusieurs fichiers et comment appeler les fonctions contenus dans chacune des fichiers.



Modules

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Maintenant que nous savons écrire et appeler des fonctions au sein d'un même fichier .py, nous allons voir comment organiser notre programme en plusieurs fichiers et comment appeler les fonctions contenus dans chacune des fichiers.

Définition

Chaque fichier .py est un module Python. Ils servent à stocker séparément des aspects différents d'un programme et peuvent être appelés dans d'autres modules afin d'accéder aux fonctions qui y sont écrites.



Espaces de noms

En Python, pour retrouver une variable ou une fonction, on se sert de son nom : c'est le **nommage des outils**. Pour retrouver un outil, on va utiliser un **espace de noms**.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Espaces de noms

En Python, pour retrouver une variable ou une fonction, on se sert de son nom : c'est le **nommage des outils**. Pour retrouver un outil, on va utiliser un **espace de noms**.

Espace de noms

Il s'agit d'un moyen de regrouper des variables et des fonctions dans un espace commun. Généralement il s'agira de leur lieu de définition.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Espaces de noms

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

En Python, pour retrouver une variable ou une fonction, on se sert de son nom : c'est le **nommage des outils**. Pour retrouver un outil, on va utiliser un **espace de noms**.

Espace de noms

Il s'agit d'un moyen de regrouper des variables et des fonctions dans un espace commun. Généralement il s'agira de leur lieu de définition.

Remarques

Il existe déjà des espaces de noms dans le domaine :

- cœur du langage pour les structures de données de base
- contexte global pour les variables d'un programme
- contexte local pour les variables d'une fonction



Espaces de noms

Ce système permet aussi d'utiliser plusieurs variables ou fonctions qui portent le même nom mais qui sont "stockées" dans des espaces de noms différents.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Espaces de noms

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Ce système permet aussi d'utiliser plusieurs variables ou fonctions qui portent le même nom mais qui sont "stockées" dans des espaces de noms différents.

Concrètement, l'identification complète d'une variable ou d'une fonction se fait de la façon suivante :

espace_de_nom.sous_espace_de_nom._nom



Espaces de noms

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Ce système permet aussi d'utiliser plusieurs variables ou fonctions qui portent le même nom mais qui sont "stockées" dans des espaces de noms différents.

Concrètement, l'identification complète d'une variable ou d'une fonction se fait de la façon suivante :

espace_de_nom.sous_espace_de_nom._nom

Remarque

La notion d'espaces de noms est liée à celle de la portée des variables.



Packages

Un programme complet peut compter des centaines de milliers de lignes de code. Il est donc indispensable de le découper en module. Cependant les modules ne seront parfois pas suffisants.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Packages

Un programme complet peut compter des centaines de milliers de lignes de code. Il est donc indispensable de le découper en module. Cependant les modules ne seront parfois pas suffisants.

Définition

Un package est un répertoire contenant un ensemble de sous-packages ou de modules (fichiers .py) ainsi qu'un fichier “__init__.py” qui peut être vide.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Packages

Un programme complet peut compter des centaines de milliers de lignes de code. Il est donc indispensable de le découper en module. Cependant les modules ne seront parfois pas suffisants.

Définition

Un package est un répertoire contenant un ensemble de sous-packages ou de modules (fichiers .py) ainsi qu'un fichier “__init__.py” qui peut être vide.

```
code/  
  geometrie/  
    __init__.py  
    systeme_coordonnees.py  
  primitive/  
    __init__.py  
    point.py  
    ligne.py  
    surface.py
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Utiliser un module

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

“Import”

Pour charger un module externe, il faut utiliser le mot clé “import” suivi du nom du module.

Une fois la commande exécutée, l’interpréteur charge dans le programme l’espace de noms du module.



Utiliser un module

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

“Import”

Pour charger un module externe, il faut utiliser le mot clé “import” suivi du nom du module.

Une fois la commande exécutée, l’interpréteur charge dans le programme l’espace de noms du module.

Pour utiliser une fonction d’un module il suffit ensuite de l’appeler : *nom_module.nom_fonction*.

```
>>> import math
>>> math.sqrt(4)
2.0
```




Différents types d'imports

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Dans la cas où l'on ne souhaite utiliser qu'une fonction du module on utilise le mot clé **from** :

```
>>> from math import sqrt
>>> sqrt(4)
2.0
```



Différents types d'imports

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules

Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Dans la cas où l'on ne souhaite utiliser qu'une fonction du module on utilise le mot clé **from** :

```
>>> from math import sqrt
>>> sqrt(4)
2.0
```

La syntaxe d'import d'un package est similaire à celle d'un module :

```
>>> import package

>>> from package.sous_package import module
```



Différents types d'imports

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

As

Lorsque l'on veut importer un module et le renommer, on peut utiliser le mot clé `as` :

```
>>> import math as m  
>>> m.sqrt(4)  
2.0
```



Différents types d'imports

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

As

Lorsque l'on veut importer un module et le renommer, on peut utiliser le mot clé `as` :

```
>>> import math as m
>>> m.sqrt(4)
2.0
```

Remarque

Pour un soucis de clarté du code, on placera les imports au début du programme, même s'ils ne sont utilisés qu'à la fin.



Modules courants

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Module	Utilisation
os	manipulation de chemins de fichiers
sys	informations sur l'environnement
time	heure courante, fuseaux horaires
datetime	gestion des dates et des durées
math	fonctions mathématiques
random	nombres aléatoires
decimal	manipulation des nombres à virgule
fractions	manipulation des fractions
doctest	documentation et tests unitaires
sqlite3	gérer des BDD sqlite3
TKinter	gérer des interfaces graphiques
matplotlib	gérer des affichages graphiques



Module “math”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
► Imports et
modules
Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Ce module introduit d'autres opérations pour les types numériques :



Module “math”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

► Imports et
modules

Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Ce module introduit d'autres opérations pour les types numériques :

- sinus, cosinus, ...
- racine carrée
- exponentielle
- ...



Module “math”

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

► Imports et
modules

Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Ce module introduit d'autres opérations pour les types numériques :

- sinus, cosinus, ...
- racine carrée
- exponentielle
- ...

```
>>> math.cos(3)          >>> math.exp(5.154)
-0.9899924966004454     173.1225975711542
```




Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 **Modules et fonctions**

- Fonctions
- Imports et modules
- Installer des packages

6 Fichiers et itérateurs

7 Conclusion



Installer des packages

Python possède beaucoup de bibliothèques open source. Il n'est pas rare au cours de l'écriture d'un programme d'avoir besoin de l'aide d'une bibliothèque externe.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Installer des packages

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python possède beaucoup de bibliothèques open source. Il n'est pas rare au cours de l'écriture d'un programme d'avoir besoin de l'aide d'une bibliothèque externe.

La méthode courante consiste à :



Installer des packages

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python possède beaucoup de bibliothèques open source. Il n'est pas rare au cours de l'écriture d'un programme d'avoir besoin de l'aide d'une bibliothèque externe.

La méthode courante consiste à :

- 1 télécharger les sources,



Installer des packages

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python possède beaucoup de bibliothèques open source. Il n'est pas rare au cours de l'écriture d'un programme d'avoir besoin de l'aide d'une bibliothèque externe.

La méthode courante consiste à :

- 1 télécharger les sources,
- 2 vérifier qu'elles contiennent bien un fichier "setup.py" ou "install.py",



Installer des packages

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python possède beaucoup de bibliothèques open source. Il n'est pas rare au cours de l'écriture d'un programme d'avoir besoin de l'aide d'une bibliothèque externe.

La méthode courante consiste à :

- 1 télécharger les sources,
- 2 vérifier qu'elles contiennent bien un fichier "setup.py" ou "install.py",
- 3 exécuter en ligne de commande la ligne suivante :

python setup.py install



Installer des packages

Problèmes

La méthode précédente pause souvent des problèmes :

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Installer des packages

Problèmes

La méthode précédente pose souvent des problèmes :

- le paquet est-il compatible avec mon architecture ? (système d'exploitation, autres paquets installés, ...)

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Installer des packages

Problèmes

La méthode précédente pose souvent des problèmes :

- le paquet est-il compatible avec mon architecture ? (système d'exploitation, autres paquets installés, ...)
- où trouver le paquet ?

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Installer des packages

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Problèmes

La méthode précédente pose souvent des problèmes :

- le paquet est-il compatible avec mon architecture ? (système d'exploitation, autres paquets installés, ...)
- où trouver le paquet ?
- comment savoir s'il s'agit de la bonne version ?



Installer des packages

Problèmes

La méthode précédente pose souvent des problèmes :

- le paquet est-il compatible avec mon architecture ? (système d'exploitation, autres paquets installés, ...)
- où trouver le paquet ?
- comment savoir s'il s'agit de la bonne version ?
- comment gérer les dépendances ? (le fait qu'un paquet en nécessite souvent d'autres pour fonctionner)

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

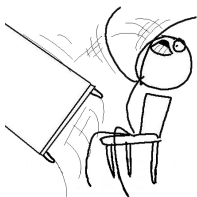


Installer des packages

Problèmes

La méthode précédente pose souvent des problèmes :

- le paquet est-il compatible avec mon architecture ? (système d'exploitation, autres paquets installés, ...)
- où trouver le paquet ?
- comment savoir s'il s'agit de la bonne version ?
- comment gérer les dépendances ? (le fait qu'un paquet en nécessite souvent d'autres pour fonctionner)





pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

```
pip3 install paquet
```



pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

```
pip3 install paquet
```

pip3 a plusieurs avantages :



pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

```
pip3 install paquet
```

pip3 a plusieurs avantages :

- il n'est plus nécessaire de télécharger le paquet



pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
▶ Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

pip3 install paquet

pip3 a plusieurs avantages :

- il n'est plus nécessaire de télécharger le paquet
- il n'est plus nécessaire de chercher les dépendances



pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
▶ Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

pip3 install paquet

pip3 a plusieurs avantages :

- il n'est plus nécessaire de télécharger le paquet
- il n'est plus nécessaire de chercher les dépendances
- il permet de choisir la version que l'on souhaite installer



pip

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules
► Installer des
packages

Fichiers et
itérateurs

Conclusion

References

Python propose depuis 2011 un gestionnaire de paquets appelé *pip3*. Il fonctionne de la façon suivante :

pip3 install paquet

pip3 a plusieurs avantages :

- il n'est plus nécessaire de télécharger le paquet
- il n'est plus nécessaire de chercher les dépendances
- il permet de choisir la version que l'on souhaite installer
- il permet de mettre à jour ou de désinstaller un paquet



Limites de pip

Lorsque l'on cherche à installer de grosses librairies (pyQt par exemple), ou lorsque l'on travaille sur des projets basés sur des versions différentes d'une librairie, ou enfin lorsque l'on souhaite tester une nouvelle librairie sans modifier son environnement, on pourra utiliser *virtualenv*.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions

Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Limites de pip

Lorsque l'on cherche à installer de grosses librairies (pyQt par exemple), ou lorsque l'on travaille sur des projets basés sur des versions différentes d'une librairie, ou enfin lorsque l'on souhaite tester une nouvelle librairie sans modifier son environnement, on pourra utiliser *virtualenv*.

virtualenv s'installe avec pip et permet de créer des espaces de travail Python séparés. On crée un environnement virtuel avec la commande :

```
virtualenv /chemin/de/l/environnement
```

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Limites de pip

Lorsque l'on cherche à installer de grosses librairies (pyQt par exemple), ou lorsque l'on travaille sur des projets basés sur des versions différentes d'une librairie, ou enfin lorsque l'on souhaite tester une nouvelle librairie sans modifier son environnement, on pourra utiliser *virtualenv*.

virtualenv s'installe avec pip et permet de créer des espaces de travail Python séparés. On crée un environnement virtuel avec la commande :

```
virtualenv /chemin/de/l/environnement
```

Pour travailler dans cet environnement, il suffit de lancer le fichier "active.bat" du répertoire. Ensuite on peut installer les paquets dont on a besoin et ils seront visibles uniquement au sein de l'environnement virtuel.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fonctions
Imports et
modules

► Installer des
packages

Fichiers et
itérateurs

Conclusion

References



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

► Fichiers et
itérateurs

Fichiers
Itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

6 Fichiers et itérateurs

7 Conclusion



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

6 Fichiers et itérateurs

- Fichiers
- Itérateurs

7 Conclusion



Fichiers

En Python, il est possible de manipuler des fichiers. Pour ce dernier type de données, nous allons voir comment lire et écrire dans un fichier.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References





Fichiers

En Python, il est possible de manipuler des fichiers. Pour ce dernier type de données, nous allons voir comment lire et écrire dans un fichier.

Attention

Pour des raisons de facilité, nous allons nous contenter de manipuler des fichiers texte, mais Python est capable de lire des fichiers binaires, des images ...

► [Verbose mode](#)



Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Chemins d'accès

Il existe 2 types de chemins d'accès différents :

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Chemins d'accès

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Il existe 2 types de chemins d'accès différents :

- chemins absolus : chemin depuis la racine du disque.
 - Windows : C:\Bureau\ENSG\PPMD\Fichier.txt
 - Linux : /home/sguinard/Bureau/Cours/Python/Fichier.txt
 - Mac OS : /Users/sguinard/Desktop/File.txt



Chemins d'accès

Il existe 2 types de chemins d'accès différents :

- chemins absolus : chemin depuis la racine du disque.
 - Windows : C:\Bureau\ENSG\PPMD\Fichier.txt
 - Linux : /home/sguinard/Bureau/Cours/Python/Fichier.txt
 - Mac OS : /Users/sguinard/Desktop/File.txt
- chemins relatifs : chemins depuis lequel l'interpréteur Python est exécuté.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Chemins d'accès

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Il existe 2 types de chemins d'accès différents :

- chemins absolus : chemin depuis la racine du disque.
 - Windows : C:\Bureau\ENSG\PPMD\Fichier.txt
 - Linux : /home/sguinard/Bureau/Cours/Python/Fichier.txt
 - Mac OS : /Users/sguinard/Desktop/File.txt
- chemins relatifs : chemins depuis lequel l'interpréteur Python est exécuté.

Pour connaître le répertoire courant, on utilise la commande :
"os.getcwd()"

```
>>> os.getcwd()  
'/home/sguinard/Bureau/Cours/Python/python_2017_2018'
```



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)
 - "r" pour lire uniquement

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)
 - "r" pour lire uniquement
 - "w" pour écrire uniquement (écrase le contenu précédent)

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)
 - "r" pour lire uniquement
 - "w" pour écrire uniquement (écrase le contenu précédent)
 - "a" pour écrire à la fin (conserve le contenu précédent)

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)
 - "r" pour lire uniquement
 - "w" pour écrire uniquement (écrase le contenu précédent)
 - "a" pour écrire à la fin (conserve le contenu précédent)

Lorsque l'on n'a plus besoin du fichier, il faut le fermer avec la méthode `fichier.close()`.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Ouverture d'un fichier

Un fichier s'ouvre avec la fonction `open()`. Cette méthode prend en paramètres :

- le chemin du fichier
- le mode d'ouverture (il est possible de combiner les modes)
 - "r" pour lire uniquement
 - "w" pour écrire uniquement (écrase le contenu précédent)
 - "a" pour écrire à la fin (conserve le contenu précédent)

Lorsque l'on n'a plus besoin du fichier, il faut le fermer avec la méthode `fichier.close()`.

Remarque

Si on cherche à ouvrir en écriture un fichier qui n'existe pas, il est automatiquement créé.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References



Lecture d'un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Maintenant que nous savons ouvrir un fichier, il faut accéder aux données à l'intérieur. Il existe plusieurs méthodes de lecture :



Lecture d'un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Maintenant que nous savons ouvrir un fichier, il faut accéder aux données à l'intérieur. Il existe plusieurs méthodes de lecture :

- “read()” : lit le fichier en entier



Lecture d'un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Maintenant que nous savons ouvrir un fichier, il faut accéder aux données à l'intérieur. Il existe plusieurs méthodes de lecture :

- “read()” : lit le fichier en entier
- “readline()” : lit une ligne du fichier



Lecture d'un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Maintenant que nous savons ouvrir un fichier, il faut accéder aux données à l'intérieur. Il existe plusieurs méthodes de lecture :

- “read()” : lit le fichier en entier
- “readline()” : lit une ligne du fichier
- certaines méthodes lisent un fichier bit par bit



Lecture d'un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Maintenant que nous savons ouvrir un fichier, il faut accéder aux données à l'intérieur. Il existe plusieurs méthodes de lecture :

- "read()" : lit le fichier en entier
- "readline()" : lit une ligne du fichier
- certaines méthodes lisent un fichier bit par bit

```
>>> file = open("fichier.txt", "r")
>>> file.read()
'Hello world'
```



Écriture dans un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Pour écrire dans un fichier, on utilise la méthode “write()”, qui prend en paramètre la chaîne de caractères à écrire. Elle renvoie le nombre de caractères écrits.



Écriture dans un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Pour écrire dans un fichier, on utilise la méthode “write()”, qui prend en paramètre la chaîne de caractères à écrire. Elle renvoie le nombre de caractères écrits.

Cette méthode n'est utilisable que si le fichier est ouvert en mode écriture (“w” ou “a”).



Écriture dans un fichier

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

Pour écrire dans un fichier, on utilise la méthode “write()”, qui prend en paramètre la chaîne de caractères à écrire. Elle renvoie le nombre de caractères écrits.

Cette méthode n'est utilisable que si le fichier est ouvert en mode écriture (“w” ou “a”).

```
>>> file = open("fichier.txt", "w")
>>> file.write("Hello world")
11
>>> file.close()
```



With

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

La manipulation de fichiers peut engendrer des erreurs, y compris en dehors du programme. Il faut donc être prudent lorsque l'on manipule des fichiers.



With

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

La manipulation de fichiers peut engendrer des erreurs, y compris en dehors du programme. Il faut donc être prudent lorsque l'on manipule des fichiers.

Pour simplifier l'utilisation des fichiers, Python possède le mot clé "with" qui garantit la fermeture automatique du fichier.



With

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Fichiers
Itérateurs

Conclusion

References

La manipulation de fichiers peut engendrer des erreurs, y compris en dehors du programme. Il faut donc être prudent lorsque l'on manipule des fichiers.

Pour simplifier l'utilisation des fichiers, Python possède le mot clé "with" qui garantit la fermeture automatique du fichier.

```
>>> with open("fichier.txt", "r") as file:  
        file.read()
```

```
'Hello world'
```




Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References

1 Présentation du module

2 Présentation de Python

3 Types et Opérations

4 Syntaxe

5 Modules et fonctions

6 Fichiers et itérateurs

- Fichiers
- Itérateurs

7 Conclusion



Itérateurs

Définition

En Python, on itère sur beaucoup d'objets : listes, chaînes de caractères, fichiers, ... Ces objets sont des itérables, c'est-à-dire qu'ils retournent des itérateurs.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References



Itérateurs

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers

► Itérateurs

Conclusion

References

Définition

En Python, on itère sur beaucoup d'objets : listes, chaînes de caractères, fichiers, ... Ces objets sont des itérables, c'est-à-dire qu'ils retournent des itérateurs.

Un itérateur est un objet ne pouvant que passer à la valeur suivante. Ils sont par exemple automatiquement utilisés dans les boucles for.



Itérateurs

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References

Définition

En Python, on itère sur beaucoup d'objets : listes, chaînes de caractères, fichiers, ... Ces objets sont des itérables, c'est-à-dire qu'ils retournent des itérateurs.

Un itérateur est un objet ne pouvant que passer à la valeur suivante. Ils sont par exemple automatiquement utilisés dans les boucles for.

Création

Un itérateur est créé à l'aide de l'opérateur "iter()".

```
>>> i = iter(dico)
>>> type(i)
<class 'dict_keyiterator'>
```



Itérateurs

Remarques

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers

► Itérateurs

Conclusion

References



Itérateurs

Remarques

- Les itérateurs ne possèdent qu'une seule méthode : "next()", qui permet de passer à l'élément suivant.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References



Itérateurs

Remarques

- Les itérateurs ne possèdent qu'une seule méthode : "next()", qui permet de passer à l'élément suivant.
- Il est possible de créer plusieurs itérateurs pour un même objet.

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References



Itérateurs

Remarques

- Les itérateurs ne possèdent qu'une seule méthode : "next()", qui permet de passer à l'élément suivant.
- Il est possible de créer plusieurs itérateurs pour un même objet.

```
>>> dico
{'bleu': '0000FF', 'vert': '00FF00', 'rouge': 'FF0000', '
    jaune': '00FFFF'}
>>> i1 = iter(dico)
>>> i2 = iter(dico)

>>> next(i1)
'bleu'
>>> next(i1)
'vert'
>>> next(i2)
'bleu'
```




Itérateurs et fichiers

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References

Les fichiers sont leurs propres itérateurs, on ne peut donc pas appeler la fonction “`iter()`” dessus.



Itérateurs et fichiers

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References

Les fichiers sont leurs propres itérateurs, on ne peut donc pas appeler la fonction “`iter()`” dessus.
Cela signifie qu’il est impossible d’avoir 2 itérateurs pour un fichier (il n’est pas possible de lire 2 lignes d’un fichier en même temps).



Boucle for

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Fichiers
► Itérateurs

Conclusion

References

Fichiers

En Python, les fichiers sont itérables avec une boucle for.

```
>>> for line in open("fichier.tct").readline():  
        print(line)
```

```
H  
e  
l  
l  
o  
  
w  
o  
r  
l  
d
```



Déroulement de la présentation

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

- 1 Présentation du module
- 2 Présentation de Python
- 3 Types et Opérations
- 4 Syntaxe
- 5 Modules et fonctions
- 6 Fichiers et itérateurs
- 7 Conclusion



Python

Python est un langage :

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,
- facile à prendre en main,



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,
- facile à prendre en main,
- plus puissant que d'autres langages interprétés (Matlab/Octave),



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,
- facile à prendre en main,
- plus puissant que d'autres langages interprétés (Matlab/Octave),
- haut niveau,



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,
- facile à prendre en main,
- plus puissant que d'autres langages interprétés (Matlab/Octave),
- haut niveau,
- ou la syntaxe est gérée par des indentations,



Python

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python est un langage :

- portable,
- facile à prendre en main,
- plus puissant que d'autres langages interprétés (Matlab/Octave),
- haut niveau,
- ou la syntaxe est gérée par des indentations,
- possédant une communauté très active,
- ...



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :

- gérer automatiquement la documentation (voir en TP),



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :

- gérer automatiquement la documentation (voir en TP),
- gérer des exceptions (voir en TP),



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :

- gérer automatiquement la documentation (voir en TP),
- gérer des exceptions (voir en TP),
- faire de la programmation orienté objet,



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :

- gérer automatiquement la documentation (voir en TP),
- gérer des exceptions (voir en TP),
- faire de la programmation orienté objet,
- gérer des bases de données,



Ce que l'on n'a pas vu

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Python permet aussi de :

- gérer automatiquement la documentation (voir en TP),
- gérer des exceptions (voir en TP),
- faire de la programmation orienté objet,
- gérer des bases de données,
- réaliser des interfaces graphiques (voir en TP)



Aller plus loin ...

Pour aller plus loin il y aussi beaucoup de librairies open source :

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References



Aller plus loin ...

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

► Conclusion

References

Pour aller plus loin il y aussi beaucoup de librairies open source :

- numpy (calcul scientifique)
- PyQt (interfaces graphiques et librairies QT)
- OpenCV (traitement d'images)
- CGAL (géométrie algorithmique)
- Django (développement web)
- Panda3D (moteur pour faire de la 3D)
- pyinstaller (création d'exécutables pour toute plateforme)



Bibliographie I

Présentation
du module

Présentation
de Python

Types et
Opérations

Syntaxe

Modules et
fonctions

Fichiers et
itérateurs

Conclusion

► Références



Internet Community. *Stack Overflow*. <https://stackoverflow.com/questions/tagged/python>. [Online; accessed 29-August-2017]. 2017.



Mark Lutz. *Programming python*. Vol. 8. O'Reilly, 1996.



Eric Matthes. *Python crash course: a hands-on, project-based introduction to programming*. No Starch Press, 2015.



Luciano Ramalho. *Fluent Python*. O'Reilly, 2015.



Guido Van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

Merci de votre attention !

Avez-vous des questions ?

mohamed.boussaha@ign.fr

victor.coindet@ensg.eu

stephane.guinard@ign.fr



Environnements de développements

Pour coder en Python, on peut écrire en terminal, mais si l'on souhaite réaliser un programme complet et réutilisable il est conseillé d'utiliser un environnement de développement.

Définition

Un environnement de développement est un logiciel qui aide à la programmation pour certains langages. Il est relié à un compilateur ou à un interpréteur et permet d'exécuter directement son code. En voici quelques exemples :

- QtCreator : C++, Java, Python
- Eclipse : C++, C#, Java, Ruby, SQL, ...
- Texmaker : \LaTeX



Environnements de développement Python

Il existe plusieurs environnements de développements utilisés avec Python :

- IDLE : environnement de développement intégré développé par Guido van Rossum en 1998
- Spyder : open source et multi-plateforme
- PyCharm : version professionnelle sous licence

▸ [Retour](#)



Plus de types !

Parmi les types non vus en cours, on peut citer :

- `unicode` : chaîne de caractères en unicode
- `basestring` : chaîne de caractères (string ou unicode)
- `frozenset` : comme un *ensemble* mais non modifiable
- `bytes` : séquence de bytes modifiable
- `bytearray` : séquences de bytes non modifiable
- `memoryview` : accéder sans copie aux données internes d'un objet
- Ellipsis (ou "...") : utilisé pour les découpes
- `NotImplemented` : objet renvoyé lors d'opérations sur des types non supportés

▸ [Retour](#)



Formatage de texte

Différences format et %

- Bug avec % et tuple (obligé d'écrire entre parenthèses et avec une virgule) et aussi avec Decimal
- % compatible avec python 2.5
- Format peut être parsé dans des appels à fonctions
- Format pas compatible avec les unicode string

▶ Retour



Fonctions anonymes

Définition

Lorsque la valeur retournée par une fonction est calculable à l'aide d'une seule instruction, Python permet de simplifier la syntaxe de la fonction avec le mot clé "lambda" :

lambda paramètres : instruction.



Fonctions anonymes

Définition

Lorsque la valeur retournée par une fonction est calculable à l'aide d'une seule instruction, Python permet de simplifier la syntaxe de la fonction avec le mot clé "lambda" :

lambda paramètres : instruction.

```
>>> produit = lambda x,y : x*y
>>> produit(3,4)
12
```



Fonctions anonymes

Définition

Lorsque la valeur retournée par une fonction est calculable à l'aide d'une seule instruction, Python permet de simplifier la syntaxe de la fonction avec le mot clé "lambda" :

lambda paramètres : instruction.

```
>>> produit = lambda x,y : x*y
>>> produit(3,4)
12
```

Remarque

À la différence des fonctions classiques, les fonctions anonymes sont des expressions et peuvent donc être employées dans une instruction sans définition préalable.

▸ Retour



Générateurs

Définition

En Python, il existe un type de fonctions spécifiques permettant de retourner un itérateur : les générateurs. Leur syntaxe est similaire aux fonctions, si ce n'est que le mot clé "return" est remplacé par le mot clé "yield".



Générateurs

Définition

En Python, il existe un type de fonctions spécifiques permettant de retourner un itérateur : les générateurs. Leur syntaxe est similaire aux fonctions, si ce n'est que le mot clé "return" est remplacé par le mot clé "yield".

```
def generateur(x):  
    n = x ** 2  
    yield n
```

```
>>> generateur(3)  
<generator object generateur at 0x7f2354ac9a98>
```

```
>>> for elem in generateur(3):  
    print(elem)
```



Générateurs

Remarques

- Un générateur peut comporter plusieurs “yield” .
- Tout ce qui peut être fait avec un générateur peut l’être avec une fonction classique.
- Les générateurs utilisent en général moins d’espace mémoire.



Générateurs

Remarques

- Un générateur peut comporter plusieurs “yield”.
- Tout ce qui peut être fait avec un générateur peut l'être avec une fonction classique.
- Les générateurs utilisent en général moins d'espace mémoire.

```
def generateur(liste):  
    for i in liste:  
        yield i ** 2  
  
def fonct_carre(liste):  
    liste2 = list()  
    for i in liste:  
        liste2.append(i ** 2)  
    return liste2
```

▸ Retour



Lecture de fichiers binaires

Pour lire un fichier en binaire, il faut l'ouvrir avec l'attribut "rb" au lieu de juste "r".

Les fonctions `read()` et `readline()` n'interpréteront pas les caractères et se contenteront de renvoyer des bytes (de la forme `b'abcdef\r\n`).

[▶ Retour](#)