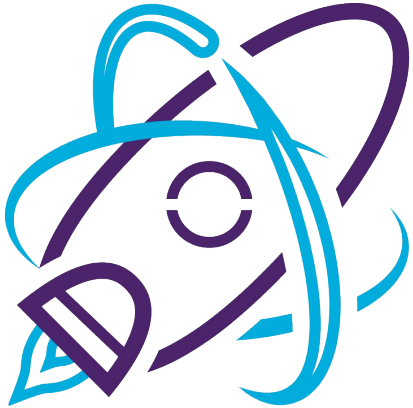


ReactIT

Formation & Recrutement
www.react-it.fr

Formation : Python

ReactIT

Votre formateur

- **Loïc Guillois**
- Développeur Web full stack depuis plus de 10 ans
- Expérience en environnement
  - SSII
  - Grand comptes Banque / Assurance / La Poste
  - Startup (Gamific.TV, La Fourchette.com, Akeneo)
  - Indépendant / freelance
- Enseignant / formateur depuis 5 ans
  - Développement
  - No SQL
  - Devops
  - Objets connectés

Faisons connaissance

Objectifs pédagogiques

- A l'issue de cette formation, vous serez en mesure de :
  - Connaître les possibilités du langage Python
  - Réaliser une application en s'appuyant sur des fonctionnalités avancées
  - Apprendre à écrire des programmes ou scripts grâce au langage Python

Prérequis du module:

  - Connaître un langage de programmation objet

Programme détaillé

Formation sur 3 jours:

- Découverte du langage
- La programmation orientée objet avec Python
- Les bibliothèques de Python
- Fonctionnalités avancées

Théorie et pratique.

Historique

- **1991** : première version du langage Python publié par son créateur (Guido Van Rossum au Pays-Bas)
- **1996** : sortie de la bibliothèque *Numerical Python* (numpy)
- **2001** : Naissance de la fondation Python (PSF : Python Software Fundation)
- Les versions se succèdent. Python est enseigné dans les université, IUT et écoles d'ingénieur. Il est utilisé en entreprise.
- **2008** : Sortie de Python 2.6 et de Python 3.0
- **2010** : versions 2.7 et 3.1.2
- **2020**: version actuelle 3.8.2

Langage open source

- Licence open source CNRI compatible GPL mais sans restriction copyleft. En conséquence, **Python est un logiciel libre et gratuit y compris pour des usages commerciaux**
- Importante communauté de développeurs
- Nombreux outils, bibliothèques et framework disponibles

Travail interactif

- **Nombreux interpréteurs interactifs disponibles**
  - ipython, jupyter…
- Importantes documentations en ligne
- Développement rapide
- Tests et débogage faciles
- Analyse interactive de données

Langage interprété rapide

- Interprétation du *bytecode* compilé
- Il est associé à un interpréteur de commandes disponible pour différents OS (Windows, Linux, MacOS X, etc.)
- De nombreux modules sont disponibles à partir de bibliothèques optimisées écrites en C, C++ ou FORTRAN

Python et la data science

- **Python est associé à de très nombreuses librairies très performantes, notamment des librairies de calcul scientifique** (Numpy, SciPy, Pandas, etc.).
- De fait, il est de plus en plus populaire, y compris auprès des data scientists.
- Il est plus généraliste que R qui est vraiment tourné vers les statistiques.
- Aujourd'hui avec l'essor de l'intelligence artificielle, il est un langage privilégié pour le machine learning

Simplicité du langage

- Syntaxe claire et cohérente
- Indentation significative
- Gestion automatique de la mémoire (garbage collector)
- Typage dynamique fort: pas de déclaration

Programmation orientée objet

- Modèle objet puissant mais pas obligatoire
- Structuration multi-fichier aisée des applications : facilite les modifications et les extensions
- Les classes, les fonctions et les méthodes sont des objets ditsde première classe. Ces objets sonttraités comme tous les autres (on peut les affecter, les passer en paramètre)

Ouverture au monde

- Interfaçable avec C/C++/FORTRAN. C'est un langage de choix pour la programmation système avec interfaçage de bas niveau
- Langage de script de plusieurs applications importantes
- Excellente portabilité

Des questions ?

Programmation Python

- **Données typées**: types usuels de la programmation : entier, réels, booléens, chaîne de caractères
- **Structures avancées de données**: Gestion des collections de valeurs (énumérations, listes) et des objets structurés(dictionnaires, classes)
- **Séquences d'instructions**: Python est un langage procédural. Il exécute les instructions les unes à la suite des autres
- **Structures algorithmiques** : Gestion des branchements conditionnels et des boucles

Programmation Python

Les outils de la programmation structurée : pouvoir regrouper du code dans des **procédures** et des **fonctions**. Cela permet de mieux organiser les applications.

Organisation du code en modules. Fichiers « `.py` » que l'on peut appeler dans d'autres programmes avec la commande `import`

Possibilité de distribution des modules : soit directement les fichiers « .py », soit sous forme d'extensions prêtes à l'emploi.

Installation de l'environnement

- Sous Windows, téléchargez Python depuis le site officiel et suivre la procédure d'installation.
- Sous Linux, installez Python par le gestionnaire de paquet. Exemple pour Ubuntu:

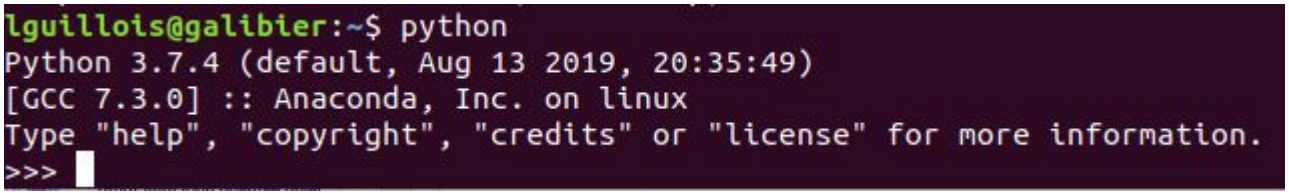
```
sudo apt-get install python3
```

Lancer l'interpréteur

Sous Windows, vous pouvez lancer l'interpréteur depuis le menu démarrer.

Sous Linux, vous pouvez le lancer par la commande suivante dans le terminal:

```
python
```



Prise en main du langage

- **Affectation, typage automatique**

```
a = 3.8
```

- **Calcul**

```
b = a + 5
```

Forcer le typage d'une variable

```
b = float(1)
```

Prise en main du langage

- **Connaître le type d'un objet**

```
type(a)
```

- **Afficher une variable**

```
print(a)
```

Supprimer un objet de la mémoire

```
del a
```

Prise en main du langage

- **Variable non défini**

```
print(a)

a + 1
```

- **Erreur de syntaxe**

```
5*2b
```

Prise en main du langage

- **Un commentaire**

```
# Ceci est un commentaire

a + 1
```

Types élémentaires de Python

- **Numérique** qui peut être `int` (entier) ou `float` (double). Les opérateurs applicables sont : `+` , `-` , `*` , `/` (division réelle), `**` (puissance) , `%` (modulo) , `//` (division entière)
- **Booléen:** `bool`, il prend deux valeurs possibles `True` et `False` (respecter la casse). Les opérateurs sont `not` (négation), `and` (ET logique), `or` (OU logique)

Exemple:

```
not(True)
```

Types élémentaires de Python

- **Chaînes de caractères**: `str`. Une constante chaîne de caractère doit être délimitée par des guillemets `"` (ou des quotes `'`)

Exemples:

```
print("Hello")

print("L'Apostrophe")

print('Hello "world"')
```

Instanciation et affectation

- **Affectation simple**

```
#typage automatique

a = 1.0

#typage explicite

a = float(1)
```

Instanciation et affectation

- **Affectations multiples**

```
# même valeur pour plusieurs variables

a = b = 2.5

# affectations parallèles

a, b = 2.5, 3.2
```

Instanciation et affectation

- La plus couramment utilisée (1 instruction = 1 ligne)

```
a = 1

b = 5

c = a + b
```

Instanciation et affectation

- **Autres possibilités**

```
a = 1;b = 5 ;c = a + b;
```

ou

```
a = 1;

b = 5;

c = a + b;
```

Transtypage

- **Principe :** utilisation du mot-clef désignant le type

```
a = "12" # a est de type chaîne caractère

b = float(a) # b est de type float
```

Peut provoquer une erreur:

```
float("hello")
```

Transtypage

- **Conversion en logique**

```
a = bool("TRUE") # a est de type bool est contient la valeur True

a = bool(1) # renvoie True également
```

- **Conversion en chaîne de caractères**

```
a = str(15) # a est de type chaîne et contient "15"
```

Opérateurs de comparaison

Sous Python, ces opérateurs sont <, <=, >, >=, != , ==

Exemple:

```
a = (15 == 13)

print(a)
```

On utilisera principalement ces opérateurs dans les branchements conditionnels.

Les entrées / sorties

- **Saisie**

```
a = input("Saisir votre valeur")

a = float(a)
```

- **Affichage**

```
print(a)
```

Branchement conditionnel

```
if condition:
    bloc d'instructions
else:
    bloc d'instructions
```

Attention à l'indentation (tabulation). La partie else est facultative.

Branchement conditionnel

- **if avec plusieurs conditions:**

```
if condition:
    bloc d'instructions
elif condition2:
    bloc d'instructions
else:
    bloc d'instructions
```

Boucles

- **Principe de la boucle:** `for` ne s'applique que sur une collection de valeurs. Ex. tuples, listes que nous verrons plus tard dans le détail.
- **Suite arithmétique simple:** On peut définir des boucles indicées en générant une collection de valeurs avec `range()`

Exemple:

```
range(4)   0 1 2 3

range(1,4)   1 2 3

range(0,5,2)   0 2 4
```

Boucles

```
for indice in séquence:

    bloc d'instructions
```

Attention à l'indentation (tabulation).

- On peut « casser » la boucle avec `break`
- On peut passer directement à l'itération suivante avec `continue`
- Des boucles imbriquées sont possibles
- Le bloc d'instructions peut contenir des conditions

Boucles

Exemple:

```
for i in range(5, 8):
    print(i, i ** 2)
print('Fin de boucle')
```

Boucles

Exemple:

```
for character in 'hello':

    print(character)
```

Boucles

Boucle `while` avec opération de comparaison, attention à la boucle infinie:

```
while condition:

    bloc d'instructions
```

Attention à l'indentation (tabulation).

- On peut « casser » la boucle avec `break`

Des questions ?

Initiation à Python

Les séquences

Définition: Une séquence est un conteneur ordonné d'éléments indexés par des entiers.

Python dispose de trois types prédéfinis de séquences :

- les chaînes (vues précédemment type `str`)
- les listes
- les tuples

Les listes

Définition: Collection ordonnée et modifiable (a.k.a. mutable) d'éléments éventuellement hétérogènes. C'est un type de tableau.

```
couleurs= ['trèfle','carreau','coeur','pique']
print(couleurs)
couleurs[1] = 14
print(couleurs)

list1= ['a','b']
list2= [4, 2.718]
list3= [list1,list2]
print(list3)
```

Les listes

Initialisation

```
t1,t2= [], [0.0] * 3
print(t1)
print(t2)

l1=list(range(4))
print("l1=",l1)
l2=list(range(4, 8))
print("l2=",l2)
l3=list(range(2, 9, 2))
print("l3=",l3)
print(2inl1, 8inl2, 6inl3)

for i in range(len(l3)):
     print(i,l3[i],sep="-",end="")
```

Les listes

Les méthodes

```
nombres = [17, 38, 10, 25, 72]
```

Tri

```
nombres.sort()
```

Ajout, suppression

```
nombres.append(12)
nombres.remove(38)
```

Les listes

Les méthodes

Inverser la liste

```
nombres.reverse()
```

Récupérer l'index d'une valeur

```
nombres.index(17)
```

Compter le nombre d'occurence

```
nombres.count(17)
```

Les listes

Manipuler les listes

Récupérer et modifier la valeur à un index donné:

```
nombres[0]
nombres[0] = 11
```

Récupérer et modifier des valeurs sur un interval d'index:

```
nombres[1:3]
nombres[1:3] = [14, 17, 2]
```

Les tuples

Définition: collection ordonnée et non modifiable (immutable) d'éléments éventuellement hétérogènes

```
mon_tuple= ('a', 2, [1, 3])
```

- Les tuples s'utilisent comme les listes mais leur parcours est plus rapide
- Ils sont utiles pour définir des constantes

Manipulation des chaînes de caractères

Quelques exemples de fonctions utiles : len (longueur), upper (passage en majuscule), find (recherche), count (compter les occurences), replace (remplacer toutes les occurences)

Exemple:

```
long = len("Hello world”)

print(upper("bonjour"))
```

Manipulation des chaînes de caractères

On peut découper une chaîne de caractère facilement.

Exemple:

```
message = "Hello world!"

print(message.split(" "))
```

Manipulation des chaînes de caractères

On peut construire une chaîne de caractère facilement.

Exemple:

```
prenoms = ['Lucien', 'Gilbert', 'Huguette']

print(" et ".join(prenoms))
```

Les dictionnaires

Les dictionnaires constituent un type composite mais ils n'appartiennent pas aux séquences.

Comme les listes, les dictionnaires sont modifiables, mais les couples enregistrés n'occupent pas un ordre immuable, leur emplacement est géré par un algorithme spécifique (hachage).

Une clé pourra être alphabétique, numérique… en fait tout type hachable. Les valeurs pourront être des valeurs numériques, des séquences, des dictionnaires, mais aussi des fonctions, des classes ou des instances.

Les dictionnaires

Définition d'un dictionnaire, exemple:

```
d1 = {'Pierre':17, 'Paul':15,'Jacques':16}

print(d1)

print(d1.items())

print(len(d1))
```

Les dictionnaires

Liste des clefs:

```
print(d1.keys())
```

Liste des valeurs:

```
print(d1.values())
```

Accès à une valeur par clé:

```
print(d1['Paul'])
print(d1.get('Paul'))
```

Les dictionnaires

Une erreur se produit si clé n'existe pas:

```
print(d1['Pipa'])
```

Modification:

```
d1['Jacques'] = 18
```

Ajout de données:

```
d1.update({'Monica':36,'Bill':49})
```

Les dictionnaires

Détecter la présence d'une clé:

```
test = 'Pierre' in d1
print(test)
```

Suppression par clé:

```
del d1['Monica']
print(d1)
```

Les dictionnaires

Les clés ne sont pas forcément des chaînes de caractères. L'outil est très souple mais, attention, autant de liberté peut être aussi préjudiciable. Il faut être très rigoureux.

Des questions ?

Les exceptions

Afin de rendre les applications plus robustes, il est nécessaire de gérer les erreurs d'exécution des parties sensibles du code.

Le mécanisme des exceptions sépare d'un côté la séquence d'instructions à exécuter lorsque tout se passe bien et, d'un autre côté, une ou plusieurs séquences d'instructions à exécuter en cas d'erreur.

Les exceptions

Lorsqu'une erreur survient, unobjet exceptionest passé au ménanisme de propagationdes exceptions, et l'exécution est transférée à la séquence de traitementad hoc.

Le mécanisme s'effectue en deux phases :

- la **levée d'exception** lors de la détection d'erreur
- le **traitement approprié**

Les exceptions

La séquence normale d'instructions est placée dans un bloc `try`.

Si une erreur est détectée (levée d'exception), elle est traitée dans le bloc `except` approprié(le gestionnaire d'exception).

Les exceptions

Exemple, les blocs else et finally sont optionnels:

```
try:
    a = 15 / 0
except ZeroDivisionError:
    print("Une erreur est survenue")
else:
    print("Bloc exécuté en l'absence d'erreur")
finally:
    print("Bloc toujours exécuté")
```

Les exceptions

L'instruction `raise` permet de lever volontairement une exception:

```
x= 2
if not(0 <=x<= 1):
    raise ValueError("x n'est pas dans[0..1]")
```

Des questions ?

Les fonctions

Une fonction est un bloc d'instruction qui prend éventuellement des paramètres en entrée et renvoie éventuellement une valeur en sortie.

Exemple:

```
def petit (a, b):
    if (a < b):
        d =a
    else:
        d = 0
    return d
```

Les fonctions

Une procédure est une fonction qui ne renvoie aucun résultat (pas de `return`)

Exemple:

```
def afficher(a):
    print(a)
```

Les fonctions

Il est possible d'affecter des valeurs par défaut:

Exemple:

```
import math
def ecart(a,b,epsilon = 0.1):
    d = math.fabs(a - b)
    if (d < epsilon):
        d = 0
    return d
```

Les fonctions

Il est possible de définir des fonctions internes:

```
def externe(a):
     def interne(b):
          return 2.0* b
     return 3.0 * interne(a)

x = 10
print(externe(x))
print(interne(x))
```

Les fonctions interne ne sont pas accessibles en dehors du bloc de définition.

Les modules

Un module est un fichier « `.py` » contenant un ensemble de variables, fonctions et classes que l'on peut importer et utiliser dans le programme principal (ou dans d'autres modules).

Le mot clé `import` permet d'importer un module.

C'est un pas supplémentaire vers la modularité : **un module maximise la réutilisation et facilite le partage du code**.

Les modules

Des modules standards prêts à l'emploi sont livrés avec la distribution Python. Exemple: random, math, os, hashlib, etc.

Voir la liste complète sur https://docs.python.org/3/library/

Exemple:

```
import math, random
random.seed(None)
value = random.random()
print(value)
```

Les modules

Exemple:

```
# définition d'alias
import math as m, random as r

# utilisation de l'alias
r.seed(None)
value = r.random()
logv = m.log(value)
abslog = m.pow(logv,2.0)
```

TP 2

Découverte du langage

Des questions ?

Restons en contact

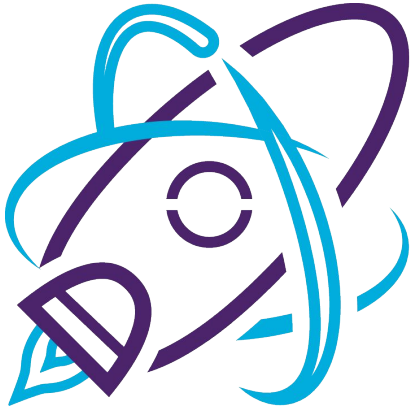
Votre contact:

- React IT
  - Loïc Guillois, président
  - hello@react-it.fr

www.react-it.fr

JUNIOR JOBS

Découvrez aussi http://junior.jobs



ReactIT

Formation & Recrutement
www.react-it.fr

Formation : Python