Introduction au python

Master IMST "Système d'information et d'adchivage numérique" (IMST-SIAN)

Fabien Rico

Univ. Claude Bernard Lyon 1

10 octobre 2022

Plan

Introduction

Le python est un langage interprété très populaire.

- Date de 1991.
- Licence libre.
- Multiplatforme, il est présent sur la plupart des systèmes.
- Propose beaucoup de bibliothèques (`Pandas`, `numpy`, etc).
- Interprété, il est lent mais facile d'utilisation et ses bibliothèques sont efficaces.
- 2e langage le plus utilisé sur github.
- Réputé simple à apprendre.

Les types

Python est un langage fortement typé

- Les variables ont un type qui est déduit par l'interpreteur
- Les opérations qui n'ont pas de sens provoquent une erreur

```
>>> "hello"+"word"
'helloword'
>>> "hello"+"1"
'hello1'
>>> "hello"+1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

- On peut définir des types grâce aux classes (langage objet)

Type de base

Python est un langage fortement typé. Les type principaux sont :

- `bool` les booléens (vrai ou faux)
- `int` les entiers (de taille arbitraire)
- `float` les nombres a virgule flottante
- `str` les chaines de caractères
- `list` les listes ou tableau
- `tuple` comme les listes mais leur taille est fixe
- `dict` les dictionnaires ou tableaux associatifs

Il est possible de définir ses propres types. Par exemple la bibliothèque pandas propose les `DataFrame` équivalent de ceux du langage `R`

Les listes

```
# définition
li = []
# ajout àla fin
li.append(1)
li.append("mot")
print(li)
# -> [1, 'mot']
# concaténation
li2 = li+li
print(li2)
# -> [1, 'mot', 1, 'mot']
```

```
# indice des éléments
# premier
print(li2[0])
# -> 1
# dernier
print(li2[-1])
#-> mot
# plusieurs à la fois
print(li2[1:3])
# -> ['mot', 1]
```

Les dictionnaires

C'est une structure de données qui associe des clefs avec des valeurs

```
# définition
d1 = {}
# ajout d'un élément
d1["nom"] = "Lagaffe"
d1["prenom"] = "Gaston"
# déclaration d'un tableau rempli
d2 = {
  "profession": "Garçon de bureau",
  "age": 65
}
# affichage
print(d1)
```

```
# Fusion de tableau
d = {**d1, **d2}
# Liste des clefs
d.keys()
#List des valeurs
d.values()
```

```
# récupération d'une valeur
d1["nom"]
d1["clefquinexistepas"] #-> génère une erreur
# avec une valeur par défaut
d1.get("clefquinexistepas", 24)
```

Branchement conditionnel `if`

```
if condition:
  bloc d'instruction
else:
  autre bloc
```

Choix du bloc à exécuter selon une condition

Exemple

```
val = int(input("Donnez un nombre svp"))
if val>=0:
     print(f"La racine carée de {val} est ")
     print(math.sqrt(val))
else:
     print(f"Impossible de calculer la racine carée de {val}")
```

Les boucles `for`

```
for indice in sequence:
  bloc d'instruction
```

Répétition d'un bloc d'instruction dans lequel l'`indice` prend chacune des valeurs de la `séquence`

Exemple

```
sum = 0
for i in range(0,10):
  sum = sum+i
  print(f"i={i}")
print(f"total={sum}")
```

Exemple

```
d={
  'nom':'Lagaffe','prenom':'Gaston',
  'profession': 'Garçon de bureau',
  'age': 65}

for k,v in d.items():
  print(f"valeur de {k} : {v}")
```

Les boucle while

```
while condition:
  Bloc d'instructions
```

Le bloc d'instructions et répété jusqu'à ce que la condition soit fausse.

Exemple

```
val = int(input("Donnez un nombre positif svp "))
while val < 0:
  val = int(input("Donnez un nombre positif svp "))
print(f"La racine carée de {val} est ")
print(math.sqrt(val))
```

Affichage

Le python permet un affichage simple avec la fonction `print`

- Sans format `print("Message à afficher")`
- Avec format `print(f"Votre valeur est {val}")`
- Ou `print("Votre valeur est {}".format(val))`

Dans ces exemple, la variable `val` est automatiquement convertie en chaine de carractères pour être insérée dans le message affiché. Attention, la conversion peut dépendre du type de la variable.
On peut lire une chaine de carractère depuis le clavier avec `input`

```
text = input("Message d'invite àafficher : ")
```

Si ce qui est désiré est différent d'une chaine de carractère, il faut le convertir

```
valentiere = int(input("Message d'invite àafficher : "))
```

Entrée/sortie dans les fichiers

On peut lire ou sauvegarder des chose dans les fichiers :

```
f = open("demofile.txt", "r")
print(f.read())
```

Le second paramètre est le mode d'ouverture, ici en lecture (*read*). Là en écriture (*write*)

```
f = open("demofile.txt", "w")
f.write("Quelquechose")
# ne pas oublier de signaler la fermeture du fichier
f.close()
```

Pour sauvegarder des données complexes, il peut etre utile de les sérialiser.

Sérialisation

La sérialisation est la transformation de données en une description qu'on peut sauvegarder dans un fichier ou transmettre sur le réseau.
En python, les plus simples à utiliser sont :

- Le *json* : condensé et très portable, mais uniquement des types simples.
- *pickle* : spécifique à python, peut sérialiser un grand nombre de choses.

Introduction au python (suite)

Master IMST "Système d'information et d'adchivage numérique" (IMST-SIAN)

Fabien Rico

Univ. Claude Bernard Lyon 1

11 octobre 2022

Plan

Mysql

Pour accéder aux base mysql, python utilise le connecteur mysql paquet `mysql-connector-python` pour pip.

```
import mysql.connector

conn = mysql.connector.connect(
  host="localhost",
  user="lenomdelutilisateur",
  password="lemotdepasse",
  database="lenomdelabase")
cursor = conn.cursor()

#... utilisation de la base

conn.close()
```

Cela permet d'exécuter toutes les requêtes, mais nous n'utiliserons que des intérrogations.

La lecture de données

Il faut générer la requete sous la forme d'une chaine de carractère.

```
import mysql.connector

conn = mysql.connector.connect(
  host="localhost",
  user="lenomdelutilisateur",
  password="lemotdepasse",
  database="lenomdelabase")
cursor = conn.cursor()

#... utilisation de la base

conn.close()
```

Cela permet d'exécuter toutes les requêtes, mais nous n'utiliserons que des intérrogations.

Faire une requete select

```
cur.execute("SELECT * FROM nomtable WHERE 1;")

# affiche les nom de colonnes (c'est un nupplet)
print(cur.column_names)
# ('colonne1', 'colonne2', ...)
results = cur.fetchall()
#
print(results)
# [(val1ligne1, val2ligne1, ...),
#  (val1ligne2, val2ligne2, ...),
#  ...
# ]
```

- On obtient chaque ligne sous la forme d'un `n-uplet` python.
- Il faut donc connaitre l'ordre des colonnes ou utiliser leur liste donnée par `cur.column_names`.
- `fetchall` permet de récuperer une liste contenant tous les résultats.
- *Une fois qu'ils ont été obtenus*, il est possible de connaitre leur nombre avec `rowcount`

Récupération au fur et à mesure

```
cur.execute("SELECT * FROM nomtable WHERE 1;")

while True:
  res = cur.fetchone()
  print(res)
  # (val1ligne_n, val2ligne_n, ...)
```

- On peut lire les résultats petit à petit.
- `fetchone()` permet de lire une ligne à la fois, `fetchmany(nb)` permet de lire par lot de `nb` résultats.
- Pourquoi est-ce intéressant ?

Ajout de parametre

La requète peut être construite à partir de variables :

```
mot = "math"
requete = f"SELECT * FROM Courses WHERE Courses.title = '{mot}'"
cur.execute(requete)
# exécution de la requete
# SELECT * FROM Courses WHERE Courses.title = 'math'
```

- La requete est une chaine de caractère, je peux utiliser les outils python pour créer celle-ci à partir de variable
- Dans l'exemple, {name} est remplacé par la valeur de la variable `name`
- Mais le forma de la requete est très contraint, que ce passe-t'il si la variable contient le carractère ' ?

```
mot = "l'experience"
requete = f"SELECT * FROM Courses WHERE Courses.title = '{mot}'"
cur.execute(requete)
# erreur la requete
# SELECT * FROM Courses WHERE Courses.title = 'l'experience'
# est invalide
```

Injection SQL

Lorsqu'un programme utilise des variables pour construire les requetes, il est possible que le contenu des variables *modifie la structure de la requete*.

- En effet, ces variable peuvent très bien contenir des caractères spéciaux de mysql.
- Si le contenu de la variable provient d'un utilisateur du programme, ce dernier peut par exemple l'écrire de manière a exécuter une autre requete.
- Cette dernière sera exécutée comme si elle provenait de votre programme.
- C'est ce qu'on appelle *l'injection SQL*
- En plus des problèmes de sécurité, les carractères spéciaux peuvent simplement déteriorer les données
  - problème de l'encodage des accents
  - erreur dans l'exécution de la requète
  - perte d'une partie de l'information

En général, rien de ce qui provient des utilisateur ou d'une source extérieure non maitrisée ne peut être transmis dans une requête sans que

Paremetre des requetes

Il existe en des fonctions pour rendre compatible les carractères spéciaux, mais ces dernier dépendent du système de base de données, de la configuration de la base ... Le plus efficace est donc d'utiliser les outils de la bibliothèque qui sont justement prévus pour.
Par exemple, en mysql, on peut utiliser des substitutions dans les requêtes :

```
name = ...
requete = f"SELECT * FROM Courses WHERE Courses.title = '%(valeur)s'"
cur.execute(requete, {"valeur": name})
```

Les autres base de données

Il y a un grand nombre de base de données. `python` propose plusieurs librairie pour les utiliser

- `Psycopg` pour les bases de données postgresql
- `cx_Oracle` pour les bases oracles
- `pyodbc` pour les différentes bases dont Microsorf sql server
- `sqlite3` pour les base sqlite

Ces librairies ont un fonctionnement similaire mais les différences rendent très difficile de maintenir un code qui fonctionnerait de manière transparente sur différents serveur.

ORM

Un ORM (*Object Relational Mapping*) est une bibliothèque qui se place entre votre programme et la base de données

- Il gère directement la sauvegarde et la récupération des objets.
- Il s'occupe de créer et modifier les tables.
- Il est capable d'utiliser plusieurs serveurs différents.
- Mais cela réduit les performances, parfois une simple requête est bien plus efficace.