PYTHON

Hubert Cardot

2017 - 2018

Enseignement

- 4h de cours et 20h TP : soit 2 CM et 10 TP

- Contrôle continu : dépôt sur Célène d'un fichier contenant vos programmes et notes (analyse…) rédigés pendant les TP (+ éventuellement une interrogation rapide sur feuille lors du dernier ou avant-dernier TP).

- Plan

- Historique et intérêt
- Syntaxe de base
- Valeurs, Variables et Affectations
- Types de données simples
- Structures de contrôle
- Types de données avancés
- Fonctions
- Classes

Historique et intérêt

- Inventé par Guido van Rossum. Première version de python est sortie en 1991
- Langage de programmation interprété: il n'est pas nécessaire de compiler avant d'exécuter
  - Portable
  - Plus lent
- Pourquoi choisir Python
  - Simple et puissant, il peut être appris en quelques jours
  - Jusqu'à 5 fois plus concis que le langage Java par exemple, ce qui augmente la productivité du développeur et réduit mécaniquement le nombre de bugs
  - Nombreuses bibliothèques : Numpy, SciPy, Pandas, Django…
  - Alternative à Matlab, Octave, Scilab, R

Ressources documentaires

- Livres
  - "Apprendre à programmer avec Python 3", Gérard Swinnen, Eyrolles
- Web
  - Page officielle de Python: http://www.python.org/
  - Un tutoriel: https://openclassrooms.com/courses/apprenez-a-programmer-en-python
  - Un autre tutoriel : http://apprendre-python.com/
  - Et un dernier : https://www.tutorialspoint.com/python/index.htm

Environnement Python

- Linux, MacOS : déjà installé. Windows : aller sur https://www.python.org/
- Choix de la version 2.7 ou 3.6
- Lancer terminal ou cmd, puis python
- Utilisation d'un éditeur : Sublime Text, Notepad++, Atom, nano…
- Ou d'un IDE : PyCharm, WingIDE, Spyder, Netbeans, Enthought…
- Voire d'un IDE en ligne : https://www.tutorialspoint.com/execute_python3_online.php

Calculs

```
>>> 1-10
-9
>>> 2*10
20
>>> 100/4
25
>>> 10%4
2
>>> 2**3
8
```

Attention

```
>>> 10/3
3.33333     en v3
3           en v2

>>> 10//3   en v3
3
```

Variables

- Typage dynamique des données : le type est défini lors de l'affection d'une valeur à la variable
- >>> a=5
  >>> a='essai'
  a change de type à la volée
- Affectations multiples et parallèles
  >>> a = b = 4
  >>> a, b = 4, 3.14
- Noms réservés :
  print in and or if del for is raise assert elif from lambda return break else global not try class except while continue exec import pass yield def finally

Type de variables

- Types de base communément utilisés:
  - entier (*int*)
  - réel (*float*)
  - chaîne de caractères (*string*)
  - Booléen (*bool*)
- Types "évolués"
  - listes, tuples, dictionnaires…
- >>> a=5
  >>> type(a)
  < class ‘int’ >

Chaînes de caractères

- Chaînes de caractères délimitées par ‘ ‘ ou « » ou « « « » » »
  caractères spéciaux précédés par \ sauf pour triple quote
- >>> ch="Eric“
  >>> print(ch[0], ch[2], ch[3])
  E i c
- >>> ch=« Bonjour » + ch
  >>> print(ch)
  Bonjour Eric
  >>> print(len(ch))
  12
- Attention: accès en ***lecture seulement***
  - une chaine de caractères est ***immutable***

Type Booléen

- Ce qui est FAUX :
  - False
  - None
  - le nombre 0 (tout type)
  - la chaîne vide
  - plus généralement, (liste, tuple, . . .) vide
- Ce qui est VRAI :
  - tout le reste, y compris True

Listes

- Comme les chaînes, les listes sont des séquences, sauf que les listes peuvent contenir n'importe quel type d'objet
- Contrairement aux chaines, les listes sont modifiables (mutables)
- Chaque objet d'une liste est accessible par l'intermédiaire d'un index (un entier qui indique l'emplacement de l'objet dans la séquence)
- >>> nombres = [5, 38, 10, 25]
  >>> print(nombres)
  [5, 38, 10, 25]
  >>> print(nombres[2]) # indice 2
  10
  >>> print(nombres[1:3]) # indices 1 à 2 inclus
  [38, 10]
  >>> print(nombres[2:]) # indices 2 à la fin
  [10, 25]
  >>> print(nombres[:2]) # du début jusqu'à l'indice 1
  [5, 38]
  >>> print(nombres[-1]) # le premier en partant de la fin
  25

Méthodes applicables aux listes

- Notation pointée (liste est une variable de type ***list***):
  - liste.sort(): trier une liste
  - liste.append(element): ajouter un élément en fin de liste
  - liste.reverse(): inverser l'ordre des éléments de la liste
  - liste.index(element): trouver l'index d'un élément
  - liste.remove(element): enlever un élément
- len(liste) : nombre d'éléments de la liste
- liste[2:2] = [element] : insertion d'un élément à l'indice 2
- del liste[2] : suppression d'un élément à l'indice 2

Copie de liste

- copieliste.py :
  # copie de listes
  liste = [1,2,3,4]
  copie1 = liste
  copie2 = liste[:]
  print(copie1, copie2)
  liste[2] = 'X'
  print(copie1, copie2)
- Donne :
  [1, 2, 3, 4] [1, 2, 3, 4]
  [1, 2, 'X', 4] [1, 2, 3, 4]
- Pour les cas plus complexes
  import copy
  copie = copy.deepcopy(liste)

Création d'une liste d'entier

- Fonction range(…)
  - Crée une séquence d'entiers
  - Peut être transformée en liste avec *list*

    list(range(10)) crée une liste des 10 premiers entiers (de 0 à 9)
- 3 utilisations possibles
  - range(N): séquence des N premiers entiers
  - range(N,M): séquence des entiers compris entre N et M
  - range(N,M,P): séquence des entiers compris entre N et M avec un pas de P

Transformation de liste en chaine

- Transformer une string en liste
- chaine = 'ceci est un essai'
  liste = chaine.split(' ') #espace
  print(liste)
- Donne : [‘ceci’, ’est’, ‘un’, ‘essai’]

- Transformer une liste en string
- chaine = " ".join(liste)

Tuples

- Un tuple est une liste non modifiable (non mutable)
- Utilisation de parenthèses à la place des crochets d'une liste
- t=(1,2,3,4)
- t=1,2,3,4 # les parenthèses sont facultatives mais aident à la lisibilité
- Si une seule valeur, ajouter une virgule : t=(1,)
- >>>t[0]
  1
- liste=list(t)
  t=tuple(liste)

Dictionnaires

- Un **dictionnaire** en **python** est une sorte de **liste** mais au lieu d'utiliser des **index**, on utilise des **clés** d'un type non modifiable (int, float, string, tuple)

- dico={}
  dico["nom"]="cardot"
  dico["prenom"]="hubert"
  print(dico)
- {'prenom': 'hubert', 'nom': 'cardot'}
- dico2 = dico.copy()

Exécution conditionnelle

- ```
  a = 2
  if a == 1:
      a = a + 1
  elif a == 2:
      a = a + 100
  elif a == 3:
      a = a + 150
  else:
      a = a - 1
  print(a)
  ```
- 102
  ```

Boucle tant que

```
a=0
while a<4:
    a=a+1
    print(a)
print('boucle tant que finie')
```

Boucle for

- nombres = [5, 38, 10, 25]
  for i in nombres:
  print(i, end = ' ')
- *i* parcourt toute la liste *nombres* et la boucle affiche tous les éléments séparés par un espace
- Parcours d'une chaine en utilisant for, range et len
- str = "radar"
  pal = True
  for i in range(len(str)//2):
  pal = pal and (str[i]==str[-1-i])
  if pal:
  print(str, "est un palindrome")
  else:
  print(str, "n'est pas un palindrome")

Fonctions

- ```
  def addition(a,b):
      return a+b
  print(addition(3,4))
  ```
- ```
  def calcul(a,b):
      return a+b, a-b
  x,y = calcul(5,3)
  print(x,y)
  ```

Passage de paramètres

- Deux explications complémentaires
  - 1. Passage par valeur d'une référence sur l'argument
    - Travail sur la copie d'une référence
  - 2. Suivant le type de l'argument
    - ***immutable*** => pas de modification possible dans la fonction
      - Types de base (int, float, string, tuple)
    - ***mutable*** => modification possible dans la fonction
      - listes, dictionnaires, objets

Exemple

- def maFonction(a,b):
  a,b = b,a
  print(a,b)
- x,y=[1,2],[3,4]
  print(x,y)
  maFonction(x,y)
  print(x,y)
- [1, 2] [3, 4]
  [3, 4] [1, 2]
  [1, 2] [3, 4]

- def maFonction(a,b):
  a[0],b[0] = b[0],a[0]
  print(a,b)
- x,y=[1,2],[3,4]
  print(x,y)
  maFonction(x,y)
  print(x,y)
- [1, 2] [3, 4]
  [3, 2] [1, 4]
  [3, 2] [1, 4]

Autres passage de paramètres

- On peut mettre des valeurs par défaut
  def f(a=0, pi=3.14):
- Ou mettre un nombre variable de paramètres
- Utilisation des variables globales :
  Si on n'utilise pas le mot-clé global, les variables globales sont utilisables en lecture mais pas en écriture.
  En la déclarant dans la fonction :
  global var_globale
  on permet sa modification.

Lambda fonction

- C’est une fonction anonyme !
  - # calcul du carré de f(z)
  - f(z) * f(z) # on calcule 2 fois f(z)
  - (lambda x : x*x)(f(z)) # on calcule une seule fois f(z)
- A laquelle on peut donner un nom :
  - carre = lambda x : x*x # on définit la fonction carre
  - carre(f(z))

Texte d’accompagnement

- ou documentation string
  ou **docstring**
- Il suit la ligne def :
  - def carré(x)
    """ Calcule l’âge du capitaine."""
    return x*x
- Permet d’obtenir :
- >>> help(carré)
  Calcule l’âge du capitaine.

Classe

- ```
  class Voiture:
      def __init__(self):
          self.nom = "Ferrari"
      def donne_moi_le_modele(self):
          return "250"

  ma_voiture = Voiture()
  print(ma_voiture.nom)
  print(ma_voiture.donne_moi_le_modele())
  ```
- ```
  Ferrari
  250
  ```

Accesseur / mutateur

- class Voiture(object):
  def __init__(self):
  self._roues=4
  def _get_roues(self):
  print "Récupération du nombre de roues«
  return self._roues
  def _set_roues(self, v):
  print "Changement du nombre de roues«
  self._roues = v
  roues=property(_get_roues, _set_roues)
- >>> ma_voiture=Voiture()
  >>> ma_voiture.roues=5
  Changement du nombre de roues
  >>> ma_voiture.roues
  Récupération du nombre de roues
  5

Destruction des Objets (Garbage Collection)

- La mémoire est récupérée quand il n’y a plus de référence sur un objet
- Python garde un compte du nombre de références pour chaque objet

Héritage de classe

- class SubClassName (ParentClass1[, ParentClass2, ...]):
  'Optional class documentation string'
  … contenu de la classe …
- Surcharge de méthodes :
  Python cherche la méthode dans la classe puis dans les classes parents dans l'ordre où ells apparaissent dans la declaration de la classe
- super() permet d'accéder directement aux méthodes des classes parents
- Toutes les classes héritent de la classe object

Modules

- Code dans un fichier .py est un script
- Commence par 2 lignes pour indiquer comment l'exécuter (sous linux) et pour prendre en compte les accents
  #!/usr/bin/python
  #-*- coding: utf-8 -*-
- from *fichier* import *fonction*
- from *fichier* import *
- from *package.fichier* import *
- Pour créer un package, il suffit de faire un répertoire avec un fichier __init__.py avec les autres fichiers .py (optionnel dans les versions récentes de python)
- Import *fichier* as *petitnom*
  Pareil que from … import … mais à l'utilisation il faut ajouter le nom du module ou le petitnom.
  module.nom_fonction ou petitnom. nom_fonction
- if __name__ == '__main__' :

Bibliothèques

- pip install nombiblio
- Utilisation des outils de l'IDE
- Utilisation de VirtualEnv

Conclusion

- Autres notions vues en TP : fichiers, interface graphique, exceptions, BD XML, calculs scientifiques…
- Nous ne verrons pas : itérateurs/générateurs, décorateurs, métaclasse…