Workshop Python

TABLE DES MATIÈRES

Premiers pas

Introduction

Qu’est-ce que Python ?

- Langage de programmation
- De script (=>interprété)
- Orienté Objet
- De (très) haut niveau

Un peu d’histoire

- 1991 : Création (Guido Van Rossum)
- Python Software Foundation (2001)
- Tyran bienveillant à vie (enfin … presque)
- Fin de python 2 !

Introduction

A quoi sert Python

- Petits scripts aux logiciels complets
- Prototypage
- Big data et machine learning
- Du web (via Django)
- De la robotique (Raspberry Poulette!)

Avantages et inconvénients

- Points +
  - Une grosse communauté
  - Un site de référence
  - Haut niveau
  - Interprété (-> multiOS)
- Points -
  - Très haut niveau
  - Interprété
  - (Pas de mobile)
  - (Desktop : Interfaces)

Opérateurs arithmétiques

Les opérateurs

- + (addition)
- - (soustraction)
- * (multiplication)
- ** (exposant)
- / (division)
- // (division entière)

Les opérateurs bien pratiques .... qui ne sont pas là

- ++ et -- (pré et post incrémentation)
- a++ doit donc a += 1

Variables

Quelques éléments de syntaxe

- pas de ; à la fin d'une instruction
- pas {} pour délimiter un bloc d'instruction -> indentation
- snake_case (évidemment :P)
- pass (instruction qui ne sert à rien : pour syntaxe)
- #commentaire monoligne

Types de données

- booléen (True ou False)
- int
- float
- string
- liste (et tuple)
- dictionnaire

Les chaines

- chaine = 'bonjour' #\exclusion et \n pour retour
- chaine = ''Bonjour'' #idem
- chaine = '''Chaine préformatée'''

Les listes => équivalent des tableaux

- ma_liste = ['bonjour', 0, True]

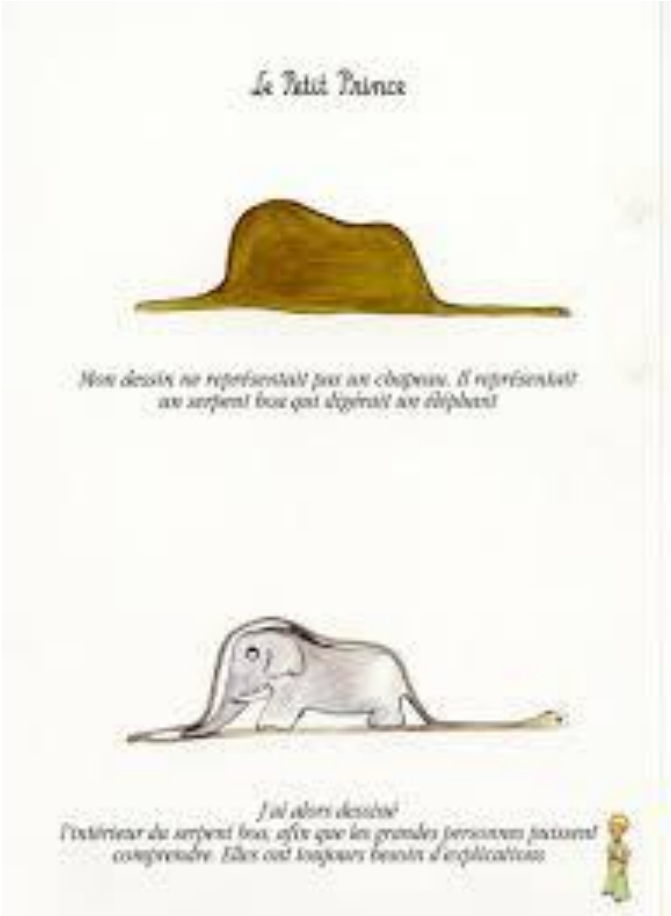
Les dictionnaires => tableaux associatifs

- mon_dico = {'clef' : valeur, 'clef2' : valeur}

Variables (Suite)

Quelques fonctions de base

- print('du texte', variable, 'encore du texte')
  - #défaut end = '\n', sep = ' '
- type(ma_variable) #renvoie son type
- entree_chaine = input('Entrez qq chose : ')
- annee = int(annee)
- -> annee = int(input('Entrez qq chose : '))



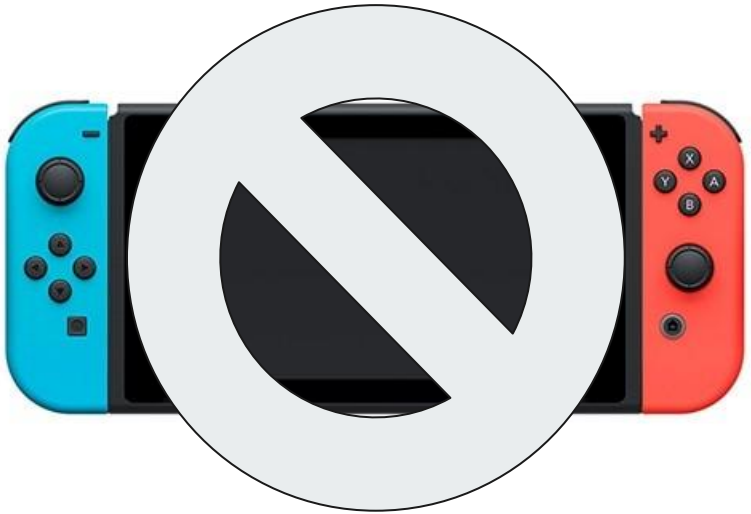
Conditions

Opérateurs de comparaison

- == et !=
- > et >=
- < et <=

Opérateurs logiques

- and
- or
- not (ex if machin is not True:)



Conditions

Structure de base d'une condition

- if condition1:
  - instructions
- elif condition2:
  - instructions
- …
- else :
  - instructions

Exemple : année bissextile ?

- annee = int(input('Entrez une annee : ')) #Gestion erreurs …
- if annee % 400 == 0 or (annee % 4 == 0 and annee % 100 != 0):
  - #code si année bissextile
- else:
  - #code si année non bissextile

Boucles

2 types de boucles

- while
- for
- **Pas de do while :(**

Modularité

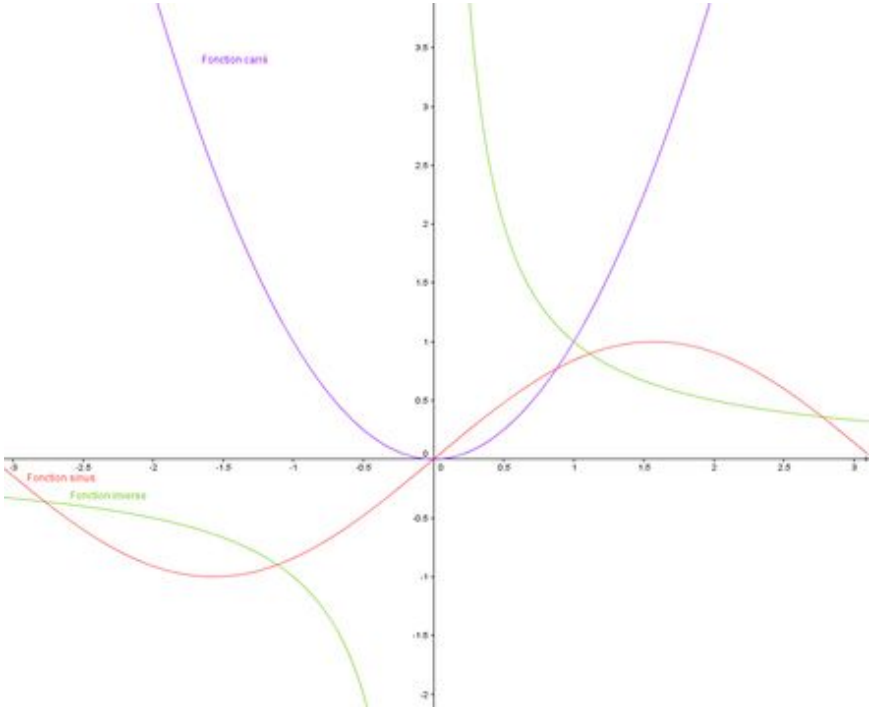
Index

- Les fonctions (et les fonctions lambda)
- Les modules
- Les packages

Les fonctions

Créer une fonction

- def nom_fonction(arg1, arg2 = valeur) :
- # arg2 = valeur -> argument facultatif



Appeler la fonction

- retour = nom_fonction(x1, ….)

Signature d’une fonction

- son nom :)

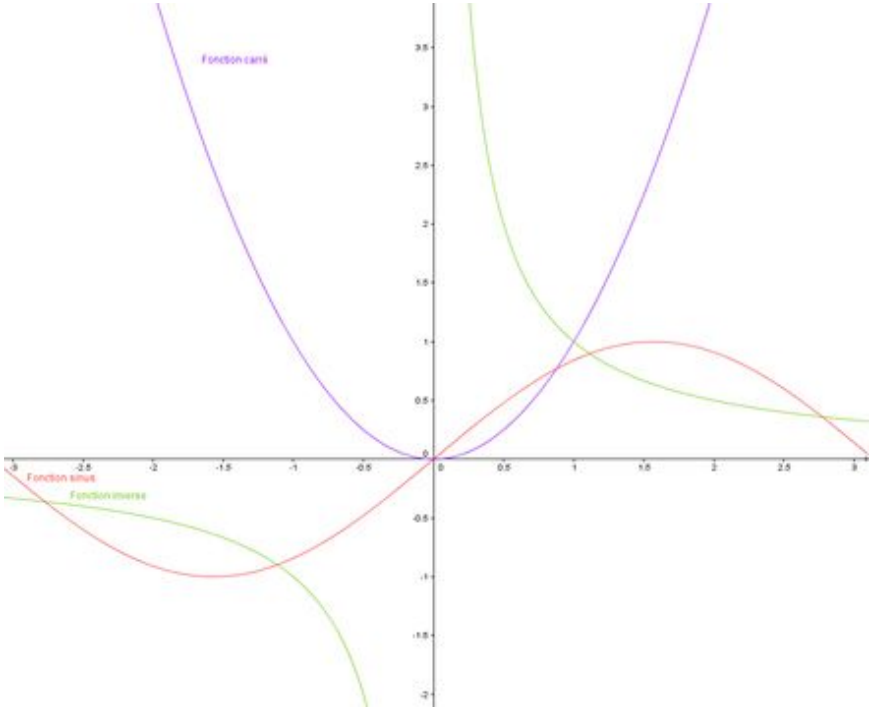
Les fonctions

Les fonctions lambda (// fonction anonyme courte)

- lambda x:x*x --- ou lambda x,y : x+y

Un exemple

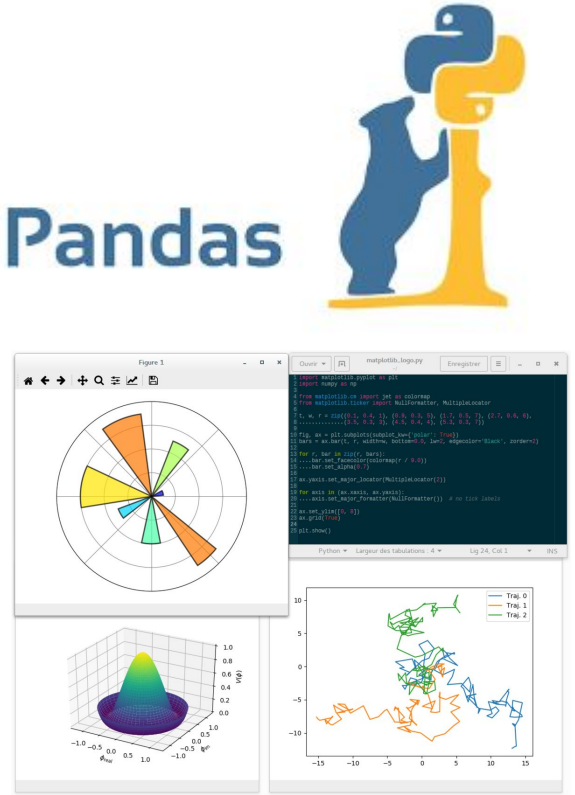
- f(5) #renvoie 25



Les modules

Index

- Importer un module
- Mettre son code dans un fichier
- Créer un module



Les modules : importer

Importer tout le module avec son namespace

- import math
- #help('math')
- math.sqrt(16) #renvoie 4
- #help("math.sqrt')

Importer une fonction module sans namespace

- from math import sqrt
- sqrt(16)

Importer tout le module en changeant son namespace

- import math as m
- #help(m)
- m.sqrt(16) #renvoie 4
- #help("m.sqrt')

Importer tout le module sans namespace (déconseillé)

- from math import *
- sqrt(16)

Les modules : mettre son code dans un fichier

Structure d'un fichier

- '''Doc du fichier'''
- # -*-coding:utf-8 -*
- import os
- Code du script
- os.system('pause')

(Bonus) : Les exceptions

Quelques erreurs

- ZeroDivisionError
- NameError
- TypeError
- AssertionError
- IndexError
- ...

Les Assertions

- assert condition
  - si condition True -> OK
  - si condition False -> AssertionError

(Bonus) : Les exceptions

Exemple

- try:
  - code à essayer
- except NameError:
  - code si NameError
- except TypeError:
  - code si TypeError
- ....
- else:
  - code si tout va bien :)
- finally:
  - code qui sera exécuté dans tous les cas

Lever une erreur (le top)

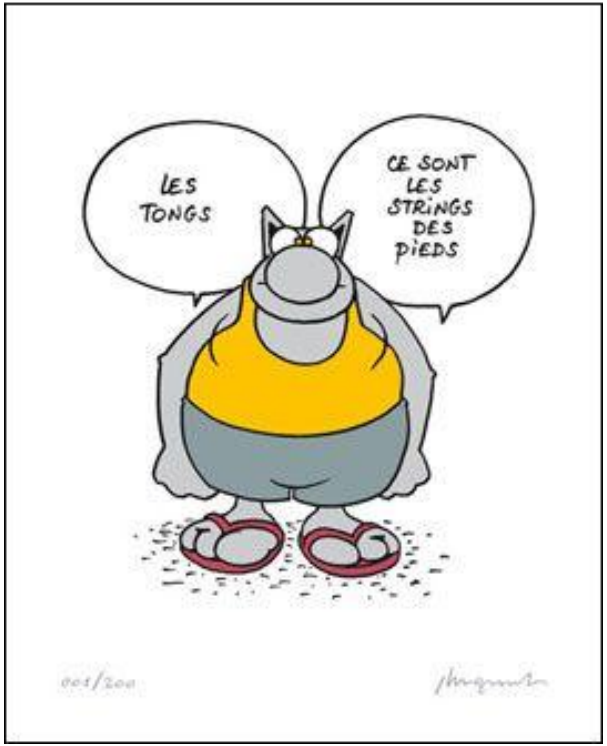
- annee = input('Entrez une annee : ')
- try:
  - annee = int(annee)
  - if annee <= 0:
    - raise ValueError("Explication")
- except ValueError:
  - #code dans ce cas

POO : Utilisation

Les chaînes

Index

- Quelques méthodes de la classe
- Concaténation
- Formater une chaine
- Parcours
- Sélection



Les chaînes : méthodes

Généralités

- Les méthodes ne modifient pas la chaine initiale
- Les méthodes renvoient donc une chaine modifiée

Concaténation

- chaine_totale = chaine1 + ‘ ‘ + chaine2
- chaine_totale = chaine1 + str(nombre) + chaine2
    - **Typage fort**

Quelques méthodes

- chaine.lower()
- chaine.capitalize()
- chaine.upper()
- chaine.strip() #== trim de js
- chaine.center(nb_caracteres)
- chaine.upper().center(nb_caracteres) #chainage de méthodes :P

Les chaînes : méthodes

Formater la chaine

- print('du texte {0} encore du texte {1}'.format(variable1, variable2)
-
- print('du texte {} et encore du texte {}'.format(variable1, variable2)
-
- print('du texte {variable1} encore du texte {variable2}'.format(variable1 = val1, variable2 = val2)

Les chaînes : parcours et sélection

Parcourir la chaine

- Par indice -> En LECTURE SEULE
  - len(chaine) → une boucle while
  - ou avec for lettre in chaine
- Quelques précisions
  - chaine[0] #première lettre
  - chaine[-1] #dernière lettre

Sélection de chaine

- chaine[0:2] #-> [0, 2[
- chaine[2:len(chaine)]
- Pour changer la chaine on doit donc passer par une chaine alternative et concaténer $
  - mot = 'bac'
  - mot = 'l' + mot[1:] -> donnera lac

Les listes => tableaux

Index

Les listes

Création

- ma_liste = list() #pas top !
- ma_liste = []
- ma_liste = [‘un’, 2 , True]

Insérer des objets dans une liste

- ma_liste.append(valeur) #ajout à la fin
- ma_liste.insert(indice, valeur)

Concaténation

- liste1.extend(liste2) #un peu verbeux …
- liste1 += liste2

Supprimer des éléments d’une liste

- del ma_liste[indice]
- ma_liste.remove(valeur) #retire la première occurrence

Les listes

Parcours de liste

- avec une boucle while et len(ma_liste)
- avec une boucle for
- *ou mieux avec la fonction enumerate !*
    - for index, valeur in enumerate(ma_liste):
        - #on peut bosser avec l'index et sa valeur associée

Entre chaine et liste

- liste = chaine.split(' ') #ou autre séparateur
- chaine = ' '.join(liste) #ou autre séparateur

Compréhension de liste

- liste = [nb*nb in liste_origine] #liste des carrés
- liste = [nb for nb in liste_origine if nb%2 == 0]
    - c'est quand mm beau hein :D
- Un petit exemple pratique
    - inventaire = [
        - ('nom_fruit', valeur),
        - ...
    -]

Un petit exemple pratique

- inventaire_inverse = [(qtt, nom_fruit) for nom_fruit,qtt in inventaire]
- inventaire = [(nom_fruit, qtt) for qtt,nom_fruit in sorted(inventaire_inverse

reverse=True)]

Les listes

Liste et paramètre de fonction (fonction dont on ne connait pas les args en avance)

- def fonction_inconnue(arg1, arg2, *commentaires):
  - #code de la fonction
- Exemple :
  - def afficher(*parametres, sep = ‘ ‘ , end = ‘\n’):
    - ‘’’Doc de la fonction’’’
    - parametres = list(parametres)
    - for i, parametre in enumerate(parametres):
      - parametres[i] = str(parametres[i])
    - chaine = sep.join(parametres)
    - chaine += fin
    - print(chaine)

Les tuples

Index

- Définition
- Création
- Cas d’utilisation implicite

Les tuples

Définition

- Sont des séquences immuables

Création

- tuple_vide = ()
- tuple_single = (1,)
- tuple_classique = (1,2,3)

Utilisations implicites

- a,b = b,a
- return a,b dans une fonction

Les dictionnaires

Index

- Créer un dictionnaire
- Ajouter des clefs-valeurs
- Supprimer des items
- Parcours
- Dictionnaire et paramètre de fonction

Les dictionnaires

Création

- mon_dictionnaire = dict()
- mon_dictionnaire = {}
- mon_dico = {‘chemises’ : 3, ‘pantalons’ : 0} oh mon dieu …

Ajouter des clefs-valeurs (ou les modifier)

- mon_dictionnaire[‘pseudo’] = ‘Arsene’
- mon_dictionnaire[‘job’] = ‘Gentleman cambrioleur’
- echiquier['a', 1] = "tour blanche" #pas que des chaines en clefs

Supprimer des items

- del mon_dictionnaire[‘job’] #les temps sont durs …
- mon_dictionnaire.pop()

Les dictionnaires

Parcours

- Parcourir les clefs
  - for clef in dictionnaire.keys():
- Parcourir les valeurs
  - for valeur in dictionnaire.values():
  - #if valeur in dictionnaire.values():
- Parcourir les 2
  - for clef, valeur in dictionnaire.items():

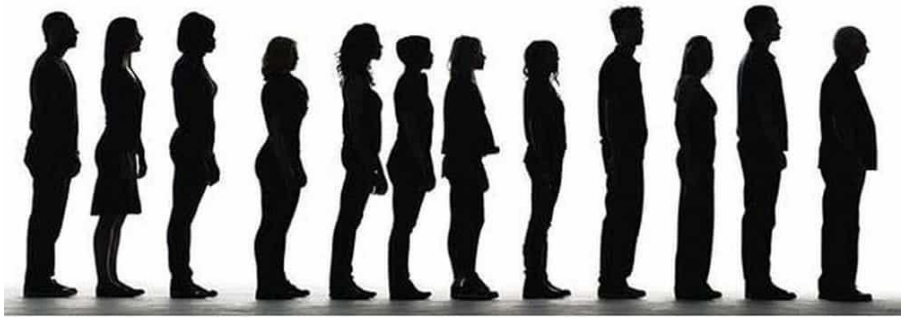
Dictionnaire et paramètre de fonction

- def fonction_inconnue(**parametres_nommes):
  - ‘’’Doc de la fonction’’’
  - print("J'ai reçu en paramètres nommés : {}.".format(parametres_nommes))
- Exactement le mm principe que pour les listes donc
  - en dernier
  - ** à la place du *

Les fichiers

Index

- ouvrir un fichier
- ferme le fichier
- lire le fichier
- écrire dans le fichier
- ouvrir et fermer en mieux
- et des objets (lecture et écriture)



One file … ^^’ srry il est tard

Les fichiers

Modes d'ouverture d'un fichier

- 'r' : lecture
- 'w' : écriture (si n'existe pas créé, si existe écrasé)
- 'a' : si n'existe pas créé, si existe ajout
- on peut aussi ajouter suffixe b (pour binary -> des objets)

Ouvrir un fichier

- mon_fichier = open('fichier.extension', 'mode_ouverture')

Fermer un fichier

- mon_fichier.close()
- mon_fichier.closed() #True ou False

Lecture d'un fichier

- mon_fichier = open('….', '..')
- contenu = mon_fichier.read() #lecture de tout le contenu
- mon_fichier.close()

Ecriture d'un fichier

- mon_fichier = open('...', '...')
- mon_fichier.write('Et voici une texte qui sera écrit') #renvoie le nb de caractères envoyés
- mon_fichier.close()

Les fichiers

ouvrir et fermer en mieux

- with open(‘fichier.extension’, ‘mode_ouverture’) as mon_fichier:
  - opération sur le fichier
  - le fichier sera fermé dans tous les cas (pratique si doit crasher)

Enregistrer un objet

- import pickle
- with open('donnees', 'wb') as fichier:
  - mon_pickler = pickle.Pickler(fichier)
  - mon_pickler.dump(mon_objet)

Récupéer un objet

- import pickle
- with open('donnees', 'rb') as fichier:
  - mon_depickler = pickle.Unpickler(fichier)
  - mon_objet = mon_depickler.load()

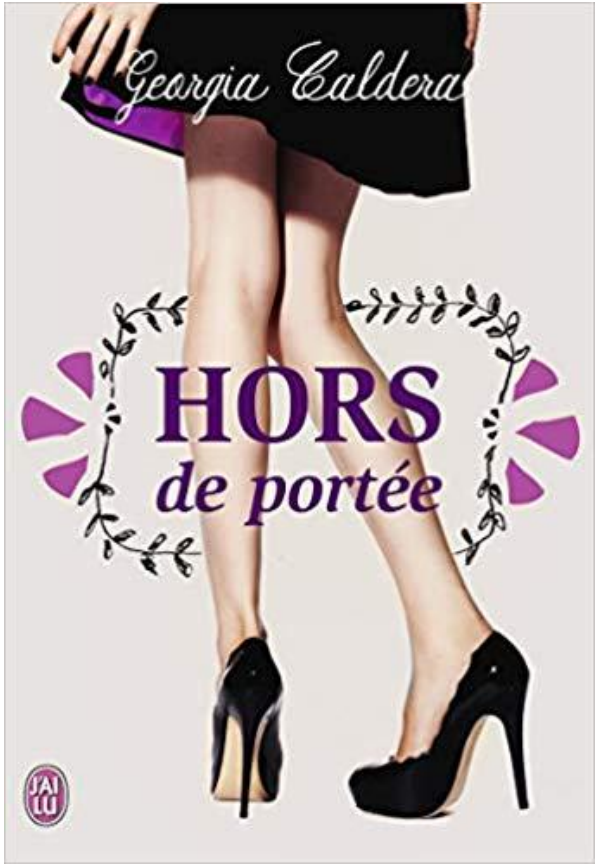
Portée et références

Références

- Comme toujours les types primitifs sont passés par valeur tandis que les types composés sont passés par référence

Portée

- Comme toujours SAUF
- Les variables globales sont par défaut en lecture seule (création d'une copie locale si on essaye un changement de valeur) si on veut modifier -> global nom_valeur

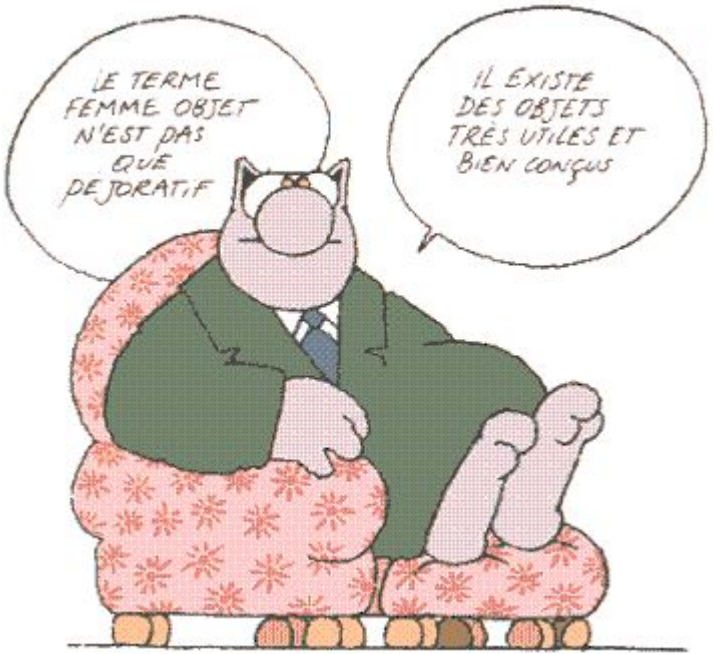


POO - Création

Principe

Index

- Attributs et méthodes
- Méthode de classe
- Méthode statique
- Introspection



Attributs et méthodes

Un exemple …

- class TableauNoir:
  - ‘’’Doc de la classe’’’
  - def __init__(self):
    - ‘’’Doc du constructeur’’’
    - this.surface = ‘’
  - def ecrire(self, message_a_ecrire):
    - ‘’’Doc de la méthode’’’
    - if(message_a_ecrire):
      - this.surface += message_a_ecrire + ‘\n’
  - def lire(self):
    - ‘’’Doc de la méthode’’’
    - return self.surface

La suite

- def effacer(self):
  - ‘’’Doc de la méthode’’’
  - this.surface = ‘’

Méthode de classe

Un exemple …

- class Compteur:
  - ‘’’Doc de la classe’’’
  - objets_crees = 0
  - def __init__(self):
    - ‘’’Doc constructeur’’’
    - Compteur.objets_crees += 1
  - def combien(cls):
    - ‘’’Doc de la méthode de classe’’’
    - return cls.objets_crees
  - combien = classmethod(combien)

Méthodes statiques

Un exemple …

- class PasInspiration:
  - ‘’’Doc de la classe’’’
  - def afficher():
    - ‘’’Doc de la méthode statique’’’
    - Code qui fait toujours mm chose (procédure) x
  - afficher = staticmethod(afficher)

Introspection

2 manières de faire

- Soit avec la fonction dir (une horreur)
- attribut spécial __dict__

Avec la fonction dir

- dir(mon_instance) #renvoie une liste avec les propriétés et méthodes y compris les héritées

Avec l'attribut spécial __dict__

- instance.__dict__ #renvoie un dictionnaire {attribut : valeur}
    - ce dico est en lecture et écriture

Les propriétés

Index

- Encapsulation ?
- Les propriétés en action

Encapsulation

- confer l'excellente veille de Olivier :p

Les propriétés en action

- property est une classe qui prend 4 paramètres optionnels
  - le getter
  - le setter
  - le deleter
  - le helper
- En pratique on ne va travailler que sur le getter , setter (pfs deleter)

Les propriétés : exemple

Exemple

- class Personne:
  - ‘’’Doc de la classe’’’
  - def __init__(self, nom, prenom):
    - ‘’’Doc constructeur’’’
    - self.nom = nom
    - self.prenom = prenom
    - self.age = 26
    - self.**_lieu_residence** = ‘Nivelles’ #_
  - def _get_lieu_residence(self):
    - ‘’’Doc du getter’’’
    - return self._lieu_residence

Suite

  - def _set_lieu_residence(self, nouvelle_adresse):
    - ‘’’Doc du setter’’’
    - self._lieu_residence = nouvelle_residence
  - lieu_residence = property(_get_lieu_residence, _set_lieu_residence)

Les méthodes spéciales

Index

- Création de l'objet (constructeur) (__init__)
- Le destructeur (__del__)
- Représentation de l'objet (__repr__, __str__)
- __getattr__ (accès à attribut qui n'existe pas ou pas accès)
- __setattr__ (mm logique)
- __delattr__ (tout dans le nom)
- etc etc (voir le fichier .odt =))

Héritage

Index

- Héritage simple
- Héritage multiple

Héritage simple

Exemple

- class B(A) :
  - ‘’’Docstring de la fonction’’’ #recherche par récursivité
  - def __init__(self, nom, matricule) :
    - ‘’’Doc du constructeur de la fille’’’
    - Personne.__ini__(self, nom) #appel du constructeur du parent pour le nom et comme ou super().__init__ dans le constructeur du parent on définit un prénom, nous en avons un maintenant
    - self.matricule = matricule

Quelques fonctions pratiques

- issubclass(fille, mere) #True ou False
- isinstance(instance, classe) #idem

Héritage multiple

Exemple

- class Fille(Mere1, Mere2) #espèce de mère2:p
  - ‘’’Docstring de la classe
  - #mm chose , juste savoir que la classe 1 a priorité sur la 2


Un pendu !